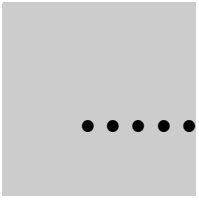


ClearTrust SecureControl

Developer's Guide

Version 4.5

July 2000

ClearTrust SecureControl, Release 4.5

Copyright © Securant Technologies, Inc. 2000

All Rights Reserved

This software/documentation contains proprietary information of Securant Technologies, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Securant Technologies, Inc. does not warrant that this document is error-free.

Securant Technologies, Inc., One Embarcadero Center, Suite 500, San Francisco, CA 94111

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com) and Tim Hudson (tjh@cryptsoft.com).

Securant, the Securant logo, and ClearTrust are registered trademarks of Securant Technologies, Inc. All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Copyright (C) 1997 Eric Young (eay@cryptsoft.com)

All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform to Netscape's SSL. This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code, not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com). Please note that MD2, MD5 and IDEA are publicly available standards that contain sample implementations, I have re-coded them in my own way but there is nothing special about those implementations. The DES library is another mater.

Copyright remains Eric Young's, and as such any copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.

This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)." The word 'cryptographic' can be left out if the routines from the Library being used are not cryptographic related.

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgment: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)."

ERIC YOUNG AS IS AND ANY EXPRESS OR PROVIDE IMPLIED WARRANTIES, INCLUDING, THIS SOFTWARE IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY

DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. I.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

The reason behind this being stated in this direct manner is past experience in code simply being copied and the attribution removed from it and then being distributed as part of other packages. This implementation was a non-trivial and unpaid effort.

Year 2000 Compliance Statement

Securant is committed to producing the highest quality software products and services. We recognize the issues and potential problems associated with the storage and calculations of dates with two digit year fields (Year 2000 problem). All dates used within our products utilize a standard four digit year or system specific representation.

When installed and implemented in accordance with our current written specifications, ClearTrust meets or exceeds the following rigorous requirements:

(i) records, stores, processes, calculates, transmits, displays, and presents calendar dates on or after (and, if applicable, spans of time including) January 1, 2000.

(ii) the use of dates before, on or after January 1, 2000 will not adversely affect performance with respect to date-dependent data, computations, output, or otherfunctions.

(iii) will not abnormally end or provide invalid or incorrect results as a result of date-dependent data; and

(iv) accurately recognize, manage, accommodate, and manipulate date-dependent data including leap years.

Chapter 1
Introduction

One key advantage of ClearTrust SecureControl as a security system is its flexibility and extensibility. Securant ClearTrust SecureControl exposes two APIs (application programming interfaces) that can be used by third-party or other developers:

- The **ClearTrust SecureControl API** enables you to implement custom functionality—usually in the area of security policy administration (batch importing of users from a proprietary directory or database, for example)—into the back-end ClearTrust servers (Authorizer, Dispatcher, Entitlements; see "Using the ClearTrust SecureControl API" elsewhere in this guide);
- The **ClearTrust Plug-in API** enables you to extend the functionality of the ClearTrust SecureControl Plug-in to meet the needs of your specific security environment. Specifically, you can extend the ClearTrust SecureControl Web Server Plug-in—using the Plug-in API to create your own Web Server Plug-in Extension, or PIX—to integrate Web Server authentication with third-party or custom authentication mechanisms not directly supported by the core ClearTrust product.

This *Developer's Guide* provides information about these topics.

Chapter 2
Using the ClearTrust SecureControl API

The ClearTrust SecureControl API enables application developers to implement custom functionality—usually in the area of security policy administration (batch importing of users from a proprietary directory or database, for example)—into the back-end ClearTrust servers. This chapter provides an overview of the API functions with information about syntax and usage. The chapter includes the following sections:

- ClearTrust ClearTrust SecureControl API Overview
- API Object Relationships
- Using the ClearTrust SecureControl API Efficiently
- ClearTrust SecureControl Object Definitions
- ClearTrust SecureControl Functions: Administrative and Runtime
- ClearTrust SecureControl Object Functions
- Relationships and Associations
- ClearTrust SecureControl API Sample Code and Examples and Examples
- Linking with the ClearTrust SecureControl C API
- Using the ClearTrust SecureControl API Across a Firewall

ClearTrust SecureControl API Overview

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

To facilitate the development of these applications, ClearTrust SecureControl provides both a Java and C API. The ClearTrust SecureControl API uses the same administrative security model as ClearTrust SecureControl Manager, limiting the scope of the functionality provided by a ClearTrust SecureControl API-enabled Application. Additionally, the ClearTrust SecureControl API simplifies auditing and problem detection through the use of server logs. For more details about the administrative security model and server logs, refer to the *ClearTrust SecureControl Policy Administration Guide*.

The ClearTrust SecureControl API provides you with many areas of functionality, some of which are discussed in the following subsections.

Batch Loading of Users

Entering new user information into the ClearTrust SecureControl GUI, one user at a time, can be time-consuming and inefficient, especially for large organizations. Using the ClearTrust SecureControl API, you can write an administrative batch-entry application to filter relevant user information from an existing database and drops it directly into ClearTrust SecureControl Entitlements database. The task of populating ClearTrust SecureControl's database is thereby reduced from many hours at the keyboard to a single invocation of a custom Application.

NOTE: Adding users programmatically may adverseley affect the Entitlements Server or API Server performance because of index setting. For large data set—over 100,000 users, for example—Securant recommends that you turn off indexing of the Entitlements database. Set the securecontrol.db.table.indexed parameter (in the SecureControl Default.conf file) to 'no.' See the Installation and Configuration Guide for more information about the Default.conf file.

Registration Tool

Some organizations may wish to give users the ability to enter or change their own personal information. The ClearTrust SecureControl API lets you write an application to modify the ClearTrust SecureControl Entitlements database. Such an application is ideal for environments with changing work conditions or high turnover.

User Management Tool

It's not always practical or expedient to change User Properties manually, one at a time—for example, when an entire division or business unit changes location. Fortunately, you can use the ClearTrust SecureControl API to create an application that filters and matches Users in the ClearTrust SecureControl database with new information from another source, and edits the information to reflect the changes. With this option you can reduce hours of repetitive data entry to a few application functions.

Application Filters

If your organization has GUI-driven applications tailored to unique business processes, you want to limit the available functions to specific Users. The ClearTrust SecureControl API allows you to filter the Application Functions available to a User based upon privilege information from the ClearTrust SecureControl database.

API Object Relationships

The ClearTrust SecureControl API gives you access to objects and their relationships. Objects are categorized as either *container* objects or *non-container* objects, and relate to other objects with a *contains*, *contained-in*, or *associated-with* relationship. Container objects are comprised of container or non-container objects, making theirs a *contains* relationships. Non-container objects are comprised only of their own defining properties (object id, name, and description) and are either *contained-in* or *associated-with* other objects.

Containment Relationships

Container objects can also have contained-in or associated-with relationships. While the *contains* and *contained-in* relationships have a fixed meaning, the *associated-with* relationship varies with the type of associated objects.

For example, the Group object is a container object comprised of User objects, but it can also be contained in a Realm object. The Group object may also be associated with an Application object. This association between the Group object and Application object defines the Group's accessibility to the Application. The association defines the relationship between the Application and any object contained in the Group.

Basic Entitlements

A Basic Entitlement specifically grants or denies a User permission to access an Application. Basic Entitlements can be granted to Users, Groups, or Realms. If granted to Groups or Realms, Basic Entitlements affect all the Users in those entities. Basic Entitlements have a distinct hierarchy; an entitlement granted or denied at the User level supersedes an opposite entitlement at the Group or Realm level. Likewise, an entitlement granted or denied at the Group level supersedes an opposite entitlement at the Realm level.

For example, if Realm A is denied access to Application X, all the Users in all the Groups in that Realm are denied access. But if Group B, a member of Realm A, is granted access, all Users in Group B are granted access. If User C, a member of Group B, is granted access to Application X, this overrides a denied entitlement at the Group level.

Multi-threaded Programming

The SecureControl C API supports connections from multi-threaded programs, such as Microsoft Transaction Server, by means of the CT_Context set of functions. If you have a single threaded application, use the original ct_ set of functions. However, if you will be developing a multi-threaded application, use the ctxt_ set of functions because it provides more control to enable multiple concurrent client connections. Both are listed in this section.

For example, there are two functions for querying what applications a user has permission to access: ctxt_get_apps_for_user() and ct_get_apps_for_user(). 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 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.)

Additionally, several return codes apply to the context-oriented function calls only, specifically:

`RC_INVALID_CONTEXT`

`RC_ALREADY_CONNECTED`

RC_NOT_CONNECTED

RC_ALREADY_INITIALIZED

The number of concurrent connections that can be managed using the context-enabled function calls is quite large-2^31 - 1.

Developers working in the Windows NT environment can use the dynamic link library (ctapi_nt.dll) and its associated static library (ctapi_nt.lib), for example, to create an application that works with Microsoft Transaction Server. Developers working in non-Windows environments can use the ctapi.lib. These files are located in the API subdirectory after installation of the ClearTrust SecureControl product.

Using the ClearTrust SecureControl API Efficiently

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

- Connect Only Once
- Minimize the Number of API Calls
- Use the User and UserProperty Functions

The following subsections describe these strategies in further detail.

Connect Only Once

The ClearTrust SecureControl API uses a persistent connection to the server; once you connect to the API Server you need not setup 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 ClearTrust SecureControl 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

The following example shows an inefficient code for displaying the names of a group of Users. The problem with this code is tht using the .size() function within the FOR loop causes a remote command to occur on the server, which will be very slow:

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

CORRECT

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 fastert than retrieving data from the server.

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

Use the User and UserProperty 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.

ClearTrust SecureControl Object Definitions

The object definitions define the various attributes of an object. Although objects may have relational attributes with other objects, these attributes are not defined within their structures.

CT_Admin Structure

A CT_Admin structure represents a ClearTrust SecureControl Administrator. A ClearTrust SecureControl Administrator is *associated with* a User. The association defines a User's administrative capabilities.

Syntax

typedef struct {

`int id;`	Reference for the API layer - do not modify!
char *name;	Name of the Administrator, used for object reference
char *description;	Textual description of the Administrator

} CT_Admin;

CT_AdminGroup Structure

A CT_AdminGroup structure represents an Administrative Group. An Administrative Group is a container of CT_Users, CT_Groups, CT_Realms, WebApplications, and CT_User Properties. The Administrative Group defines which Administrators *own* (modify) a set of objects.

All Administrators contained within an Administrative Group have *ownership* of all objects (User, Groups, and so on) within an Administrative Group.

A CT_AdminGroup can be associated with any number of Administrator structures. Each of the Administrator structures defines a set of administrative functions.

An Administrator is also associated with a User. This set of relationships gives a specific User a set of administrative duties to perform, and a set of objects to administer or own.

NOTE: The API only gives access to the AdminGroup's descriptive information and it contains relationships. The API will not allow you to create or delete an AdminGroup object, nor will it allow yo to set or delete an AdminGroup's associated-with relationship to an Administrator. To perform these functions, you must use the ClearTrust SecureControl GUI.

Syntax

typedef struct {

`int i`d;	Reference for the API layer - do not modify!
char *name;	Name of the Administrative Group
char *description;	Textual description of the AdminGroup

char *password policy;Name of the Password Policy associated with this Administrative Group.

`CT_BOOLEAN defaultPrivate`;If TRUE, Users created in this Administrative Group the GUI are private or public be default.

`CT_BOOLEAN forceExpiry`;If TRUE, Users created in this Administrative Group have expired passwords on creation. On first login, the Users will be required to change their password.

} CT_AdminGroup;

CT_Application Structure

A CT_Application structure represents a container of URLs. An Application is associated with a User, Group, or Realm to define the Application's accessibility.

Note: *The Application structure does not describe the Application's* associated-with *relationships. The Application structure contains only descriptive information.*

Syntax

typdef struct {

`int id;`	*Reference for the API layer - do not modify!*
`char *name;`	Name of the Application, used for obj reference and display label on the gui
`char *version;`	*Version of the Application*
`char *description;`	*Textual description of the Application*

`CT_BOOLEAN ct_public;` *If TRUE, the object is modifiable by any Administrator.*

} CT_Application;

CT_ApplicationFunction Structure

A CT_ApplicationFunction structure represents a function that can be performed on a given application.

Syntax

typdef struct {

`int id;`	Reference for the API layer - do not modify!
char *name;	Name of the function, used for obj reference and display label on the gui

char *description; Textual description of the function

CT_BOOLEAN filter; TRUE allows access when two Smart Rules conflict. FALSE denies access when two Smart Rules conflict.

} CT_ApplicationFunction;

CT_ApplicationURL

A CT_ApplicationURL structure represents a URL contained in an Application.

NOTE: The ApplicationURL structure does not describe the ApplicationURL's associated-with relationships. The Application structure contains only descriptive information.

Syntax

typedef struct {

`int id;` *Reference for the API layer - do not modify!*

`int app_id;` *Id of the URL's Web App*

`int websrvr_id;` *Id of the URL's Web Server*

`char *description;` *Textual description of the ApplicationURL*

`char *uri;` *The accessible file. Note: This does not include the protocol (e.g. http*) or the host name*

} CT_ApplicationURL;

CT_ExplicitEntitlement Structure

A CT_ExplicitEntitlement defines a CT_User, CT_Group, or CT_Realm's accessibility to a CT_Application.

NOTE: In order to create a Basic Entitlement, you must specify its Entity id and its Application id.

Syntax

typedef struct {

`int id;` *Reference for the API layer - do not modify!*

`CT_BOOLEAN accessible;` *Defines whether the Application is accessible*

} CT_ExplicitEntitlement;

CT_EntityHdr Structure

The CT_Entity Header contains common attributes for CT_Users, CT_Groups, and CT_Realms.

Syntax

typedef struct {

`int id;` *Reference for the API layer - do not modify!*

`char *name` *The name of the entity*

`CT_BOOLEAN ct_public;`*If an entity is modifiable by any Administrator*

} CT_EntityHdr;

CT_Group Structure

A CT_Group structure represents a group of Users.

Note: *The CT_Group structure does not describe the relationship between the Group and its Users. The Group structure contains only descriptive information of the Group object.*

Syntax

typedef struct {

`CT_EntityHdr hdr;`*Common entity information*

`char *description;` *Textual description of the group*

} Group;

CT_Realm Structure

A CT_Realm structure represents a group of Groups.

Note: *The Realm structure does not describe the relationship between the Realm and its Groups. The Realm structure contains only descriptive information.*

Syntax

typedef struct {

`CT_EntityHdr hdr;`*Common entity information*

char *description; *Textual description of the Realm*

} **CT_Realm;**

CT_User Structure

A CT_User structure represents either a ClearTrust SecureControl Administrator or a User of a ClearTrust SecureControl-protected system.

Syntax

typedef struct {

CT_EntityHdr hdr;*Common entity information*

char *firstname; *First name of User*

char *lastname; *Last name of User*

char *emailaddr;*Email address of User*

CT_BOOLEAN superuser; *If true, User is an SecureControl Super Administrator*

char *password;*Password - password of the User, used only to set the password, it isn't set when a User is retrieved from the server*

char *dn; *Distinguished Name of a User. This attribute is used to map a SecureControl User to a User in an external directory*

ObjectArrayRef propArray;

Array of User's Properties

} CT_User;

Object Array Reference

If the User was retrieved using the **get_user_and_properties** function, the User structure contains an array of properties. The Object Array Reference is as follows:

typedef struct {

ObjectRef obj_ref; *Reference to the object array*

int num_of_objs; *Number of objects in the array*

} ObjectArrayRef;

typedef struct {

OBJ_TYPE obj_type; *The type of object*

void *obj_ptr; *Pointer to an object*

} ObjectRef;

The ObjectArrayRef contains an array of ObjectRef structures; the array is contained in the **obj_ref** variable. The **num_of_objs** variable specifies the number of ObjectRef structures.

The ObjectRef structure contains an object type, **obj_type**, which identifies the type of object in the ObjectRef structure. The UserProperty array is referenced by **obj_ptr**.

CT_UserProperty Structure

A CT_UserProperty structure describes a particular property of a User.

The CT_UserProperty's value is self-defining and is used by Smart Rules to calculate a User's accessibility to an Application.

Note: *Smart Rules are defined via ClearTrust SecureControl's GUI and are not accessible through the ClearTrust SecureControl API.*

Syntax

typedef struct {

int user_id; *Reference for the API layer - do not modify!*

char *name; *Name of UserProperty, used for object reference*

USERPROPTYPE type; *Defines the data type of the User Property*

USERPROPVALUE val; *Value of the data type*

char *description; *Textual description of the User Property*

} **CT_UserProperty**

How to Set User Properties

```
time_t cur_time = time(NULL);
DATE_TYPE start_date = localtime(&cur_time);
DATE_TYPE end_date = start_date;
CT_User user;
CT_UserProperty user_props {2};

memset(&user, '\0', sizeof(CT_User));
memset(&user_props, '0\', 2 * sizeof(CT_UserProperty));
end_date->tm_year++
user.hdr.name = "ireilly";
user.hdr.ct_public = TRUE;
user.firstname = "Ignagius";
user.lastname = "Reilly";
user.emailaddr = ireilly@llevypants.com;
user.superuser = FALSE;
user.superHelpDesk = FALSE
user.password = "password";
user.dn = NULL;
user.is.locked = FALSE
user.startdate = *start_date;
user.enddate = *end_date;

user_props[0].name = "Employee Serial #";
user_props[0].type = INT_VALUE;
user_props[0].val.intval = 612142;
user_props[1].name = "Social Security #";
user_props[1].type = STRING_VALUE;
```

```
user_props[1].val = “123-45-6789”;


return_ct_create_user_and_properties(&user, 2, &user_props)
```

How to Retrieve User Properties

```
CT_User *user;
if (rc ==ct_get_user_and_properties(“irielly”, &user))
{
{
CT_UserProperty *user_props = (CT_UserProperty *) user-
>propArray.obj_ref.obj_ptr;
   printf(“Employee Serial # =%d\n”,
user_props[0].val.intval);
   printf(“Social Security # =%s\n”,
user_props[1].val.strval);


}
```

CT_WebServer Structure

A CT_WebServer structure represents a Web server.

A WebServer defines a URL’s location through an *associated-with* relationship. Also, the WebServer object represents the location of a ClearTrust SecureControl authorizer which performs accessibility checking against the associated URLs.

Note: *The WebServer structure does not include the WebServer’s associated-with relationships. The WebServer structure contains only descriptive information.*

Syntax

typedef struct {

`int id;`	*Reference for the API layer - do not modify!*
`char *name;`	*Name of the WebServeri*
`char *description;`	*Textual description of the WebServer*

`char *manufacturer;` *Manufacturer of the WebServer*

`char *hostname;` *Host name of the WebServer*

`int port;` *Port number of the WebServer*

`CT_BOOLEAN ct_public` *Defines the CT_WebServer's visibility to a SecureControl CT_WebServer.*

} CT_WebServer;

SecureControl Functions: Administrative and Runtime

The ClearTrust SecureControl API includes two types of functions:

- Administrative Functions, which enable you to programmatically modify the Entitlements database, and therefore, require the developer to logon with an appropriate ClearTrust Adminitrator's account name and password;
- Runtime Functions, which enable you only to query the Entitlements database settings and status. You don't need administrative privileges to do this; you can simply log on using NULL as the User name and password and carry out any of the functions in that category.

Administrative and Runtime functions are described on the following pages. Both types of functions support connections from multi-threaded programs, as discussed briefly in the next section.

Administrative Functions

Administrative functions interact with the Entitlements Database to modify the SecureControl data model. This section describes the administrative functions.

ctxt_connect (and ct_connect)

```
int ctxt_connect(CT_Context *context, char *srvr_name, char
*user, char *pw, char *role, char *port, int use_ssl);
```

The **ctxt_connect** function calls ctxt_connect_admin internally, with admin_group set to NULL. (The `ct_connect` function will be deprecated in a future release of the API. It is only here to maintain reverse compatibility.)

Input

context	*And this is???*
srvr_name	*The server's hostname or ip address*
user	*Name of the Administrator*
pw	*Administrator's password*
role	*Name of Administrative Role*
port	*The server's port*

NOTE: If the User and password are Null, only the socket is established.

Outputnone

Return Code

RC_OK	*Successfully retrieved the number of objects*
RC_INCORRECT_PW	*Incorrect Password*
RC_OBJ_NOT_FOUND	*Object not found*
RC_USER_NOT_DEFINED	*User not defined.*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

The default API port is 5601

Syntax

```
int ct_connect(char *srvr_name, char *user_name, char *pw,
char *admin_name, char *port, int use_ssl);
```

NOTE: To avoid unexpected system behavior, make sure the **use_ssl** setting matches the setting in the Entitlements Server. Refer to the section titled, Authorization Server Configuration Definitions in Chapter 1 of the Installation and Configuration Guide for information on configuring the Default.conf file of the Entitlements Server.

ctxt_connect_admin (and ct_connect_admin)

Connects to the SecureControl API Server. If the admin_user or admin_pw is NULL, only the socket is established.

Input

context	*Pointer to the CT_Context token.*
srvr_name	*The server's hostname or ip address*
port	*API Server's port*
use_ssl	*A boolean value representing what*
admin_user	*Name of the Administrative User (optional)*
admin_pw	*Administrative User's password (optional)*
admin_role	*Name of the Administrative Role (optional)*
admin_group	*Name of the Administrative Group (optional)*

Output*none*

Return Code

RC_OK	*Successfully retrieved the number of objects*
RC_INCORRECT_PW	*Incorrect Password*
RC_USER_NOT_DEFINED	*User not defined*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*
RC_MEMORY_ERROR	*Memory Error*
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

Syntax

```
int ctxt_connect_admin(CT_Context *context, char *srvr_name,
char *port, int use_ssl, char *admin_user, char *admin_pw,
char *admin_role, char *admin_group);
```

```
int ct_connect_admin(char *srvr_name,  char *port, int
use_ssl, char *admin_user, char *admin_pw, char *admin_role,
char *admin_group);
```

ctxt_create_entitlement (and ct_create_entitlement)

This function creates a Basic Entitlement between an Entity (User, Group, or Realm) and a Web Application.

Input

obj_type	*Type of the entity (CT_USER, CT_GROUP, or CT_REALM)*
obj_ptr	*Pointer to the entity object*
webapp_ptr	*Pointer to the Web Application*
func_name_ptr	*Pointer to the Application Function name, for access, the name is ACCESS*

Output

ent_ptr	*Pointer to the returned explicit entitlement*

Return Code

RC_OK	*Successfully created entitlement*
RC_INVALID_TYPE	*Specified invalid object type*
RC_OBJ_NOT_FOUND	*Object not found*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

Syntax

```
int ctxt_create_entitlement(CT_Context context, OBJ_TYPE
obj_type, void *obj_ptr, CT_Application *webapp_ptr,
CT_ExplicitEntitlement *ent_ptr, char *func_name);


int ct_create_entitlement(OBJ_TYPE obj_type, void *obj_ptr,
CT_Application *webapp_ptr, CT_ExplicitEntitlement *ent_ptr,
char *func_name);
```

ct_create_user_and_properties

This function creates a User and all of its properties. This function provides an efficient way to store a new User and all of its properties in one single call. You must be logged in as a Super User to use this function.

Input

`user_ptr`	*Pointer to a User*
`array_size`	*Number of arrau elements*
`props_ptr`	*Pointer to an array of User Properties*

Output

`user_ptr`	*Pointer to a User*
	This propArray field points to the array of UserProperties

Return Code

`RC_OK`	*Successfully retrieved entitlement*
`RC_INVALID_TYPE`	*Specified invalid object type*
`RC_DUPLICATE_OBJ_ERROR`	*A User with the same name already exists*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_create_user_and_properties(CT_User *user_ptr, int array_size, CT_UserProperty *props_ptr);

ctxt_disconnect (and ct_disconnect)

This function disconnects you from the ClearTrust SecureControl API Server.

Inputnone

Outputnone

Return Code

`RC_OK`	*Successfully retrieved the number of objects*
`RC_NOT_CONNECTED`	*User is not connected*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

```
int ctxt_disconnect(CT_Context context);


int ct_disconnect();
```

ct_flush_cache

This function flushes the ClearTrust SecureControl Authorizer's caches in order to notify the Authorizer of all changes in the Entitlements Database.

Inputnone

Outputnone

Return Code

`RC_OK`	*Cache was flushed*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_flush_cache();

ct_force_password_expiration

This function forces the expiration of a specified User's password. This forces the User to change his/her password the next time he/she authenticates.

Input

`user_name`	*Pointer to a User's name or Distinguished Name (DN)*

force_expiration*Bit indicating whether or not the User's password will be forced to expire*

Outputnone

Return Code

RC_OK	*User was found and the password forced expiration was set*
RC_OBJ_NOT_FOUND	*User not found*
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

Syntax

ct-force-password-expiration (char *user_name);

NOTE: The default password, ch4nge_me is automatically set when you create a new User through the ClearTrust SecureControl Manager. This password is inactive, however, if the force_password_expiration parameter is set to TRUE. If this parameter is NOTset to TRUE, the password is active for the period of time specified in the password lifetime of the Password Policy associated with the Administrative Group that created the User.

ctxt_get_entitlement (and ct_get_entitlement)

This function retrieves a Basic Entitlement between an Entity (User, Group, or Realm) and a Web Application.

Input

context	*CT_Context token*
obj_type	*Type of the entity (USER, GROUP, or REALM)*
obj_ptr	*Pointer to the entity object*
webapp_ptr	*Pointer to the Web Application.*
func_name_ptr	*Pointer to the Application Function name, for access, the name is ACCESS*

Output

ent_ptr	*Pointer to the returned basic entitlement*

Return Code

`RC_OK` *Successfully retrieved entitlement*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_OBJ_NOT_FOUND` *Object not found*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

```
int ctxt_get_entitlement(CT_Context context, OBJ_TYPE
obj_type, void *obj_ptr, CT_Application *webapp_ptr,
CT_ExplicitEntitlement **ent_ptr, char *func_name);


int ct_get_entitlement(OBJ_TYPE obj_type, void *obj_ptr,
CT_Application *webapp_ptr, CT_ExplicitEntitlement
**ent_ptr, char *func_name);
```

ct_get_user_and_properties

This is a convenience function for retrieving a User and all of its properties.

Input

`user_name` *Pointer to a User's name*

Output

`user_ptr` *Pointer to a User*

The propArray field points to array of UserProperties

Return Code

`RC_OK` *Successfully retrieved entitlement*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_OBJ_NOT_FOUND` *Object not found*

RC_NOT_AUTHORIZED *User not authorized to perform the function*

RC_TRANSPORT_ERROR *Error attempting to communicate with the server*

Syntax

int ct_get_user_and_properties(char *user_name, CT_User **user_ptr);

ct_get_user_and_properties_by_dn

This is a convenience function for retrieving a User and all of its properties by DN (Distinguished Name).

Input

dn *Pointer to a User's Distinguished Name*

Output

user_ptr *Pointer to a User*

The propArray field points to the array of UserProperties

Return Code

RC_OK *Successfully retrieved entitlement*

RC_INVALID_TYPE *Specified invalid object type*

RC_OBJ_NOT_FOUND *Object not found*

RC_NOT_AUTHORIZED *User not authorized to perform the function*

RC_TRANSPORT_ERROR *Error attempting to communicate with the server*

Syntax

int ct_get_user_and_properties_by_dn(char *dn, CT_User **user_ptr);

ctxt_reset_password (and ct_reset_password)

This function resets a User's password.

Input

context *Pointer to CT_Context token*

user_name *Pointer to a User's name*

old_password *Pointer to the User's old password*

new_password *Pointer to the User's new password*

Outputnone

Return Code

RC_OK *Successfully set the password*

RC_INVALID_PASSWORD*The new password supplied violated the server's password policy*

RC_OBJ_NOT_FOUND*Object not found*

RC_NOT_AUTHORIZED*User not authorized to perform the function*

RC_TRANSPORT_ERROR*Error attempting to communicate with the server*

Syntax

```
int ct_reset_password(char *user_name, char *old_password,
char *new_password);


int ctxt_reset_password(CT_Context context, char *user_name,
char *old_password, char *new_password);
```

Note: *You do not need to supply an Administrative name, password, or Role on the connect statement to use this function. This allows you to write a CGI or application that does not store the ClearTrust SecureControl Administrator's ID or password. To use this function successfully, you must supply the User's old password.*

ct_save_user_and_properties

This function saves a User and all of its properties. This provides for an efficient way of storing a User and all of its properties in one single call. If the definition for a UserProperty doesn't exist, this call creates a new definition for the property. You must be logged on as a Super User to use this function.

Input

user_ptr	*Pointer to a User*
array_size	*Number of array elements*
props_ptr	*Pointer to an array of User Properties*

Outputnone

Return Code

RC_OK	*Successfully retrieved entitlement*
RC_INVALID_TYPE	*Specified invalid object type*
RC_OBJ_NOT_FOUND	*Object not found*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

Syntax

int ct_save_user_and_properties(CT_User *user_ptr, int array_size, CT_UserProperty *props_ptr);

ctxt_set_password (and ct_set_password)

This function sets a User's password.

Input

user_ptr	*Pointer to a User object*
password	*Pointer to the User's password*

Outputnone

Return Code

RC_OK	*Successfully set the User's Password*
RC_OBJ_NOT_FOUND	*Object not found*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

```
int ctxt_set_password(CT_Context context, char *user_name,
char *password);


int ct_set_password(CT_User *user_ptr, char *password);
```

ClearTrust SecureControl Object Functions

This section discusses ClearTrust SecureControl API functions that manipulate the following stand-alone objects:

- CT_REALM
- CT_GROUP
- CT_USER
- CT_USER_PROPERTY
- CT_USER_PROPERTY_DEF
- CT_ADMINISTRATIVE_GROUP
- CT_ADMINISTRATOR
- CT_APPLICATION_URL
- CT_WEB_SERVER
- CT_APPLICATION_FUNCTION
- CT_EXPLICIT_ENTITLEMENT

ct_alloc_obj

The **`ct_alloc_obj function`** allocates a local ClearTrust SecureControl object. Use this function call instead of allocating storage directly.

Input

`obj_type` *The object type to allocate*

`obj_ptr` *Pointer to the object to allocate*

Valid Object Types

CT_REALM

CT_GROUP

CT_USER

CT_USER_PROPERTY

CT_ADMINISTRATIVE_GROUP

CT_ADMINISTRATOR

CT_APPLICATION

CT_APPLICATION_URL

CT_WEB_SERVER

CT_EXPLICIT_ENTITLEMENT

CT_APPLICATION_FUNCTION

CT_USER_PROPERTY_DEF

Output

`obj_ptr` *Pointer to allocated storage*

Return Code

`RC_OK` *Successfully allocated the object*

`RC_INVALID_TYPE` *Invalid object type*

`RC_MEMORY_ERROR` *Either the client or server is out of memory*

Syntax

int ct_alloc_obj(OBJ_TYPE obj_type, void **obj_ptr);

ct_create_obj

This function creates an object of a given object type.

Note: *Not all object types can be created. The allowed object types are: Group, Realm, User, User Property, Application, Web Application, URI, and Web Server.*

Input

`obj_type` *The object type to create*

Output

`obj_ptr` *Pointer to the new object with its initial values*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_USER_PROPERTY

CT_APPLICATION

CT_APPLICATION_URL

CT_WEB_SERVER

CT_USER_PROPERTY_DEF

Output

`obj_ptr` *Pointer to the new object*

Return Code

`RC_OK` *Successfully created the object*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_create_obj(OBJ_TYPE obj_type, void *obj_ptr);

ct_get_num_of_objs

Given the object type, this function returns the number of objects in the Entitlement Server database.

Input

`obj_type` *The object type to query*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_APPLICATION

CT_ADMINISTRATIVE_GROUP

CT_WEB_SERVER

CT_USER_PROPERTY_DEF

Output

`num_of_objs` *Contains the number of objects for the specified type*

Return Code

`RC_OK` *Successfully retrieved the number of objects*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

RC_MEMORY ERROR*Either the client or server is out of memory*

`RC_TRANSPORT_ERROR`*Error attempting to communicate with the server*

Syntax

int ct_get_num_of_objs(OBJ_TYPE obj_type, int *num_of_objs);

ct_get_obj_by_index

Given the object type and object index, this function returns the object.

Input

`obj_type` *The object type to query*

`obj_index` *Index of the object*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_APPLICATION

CT_ADMINISTRATIVE_GROUP

CT_WEB_SERVER

CT_USER_PROPERTY_DEF

Output

`obj_ptr` *Pointer to the requested object*

Return Code

`RC_OK` *Successfully retrieved the object*

`RC_OBJ_NOT_FOUND`*Object not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

RC_MEMORY ERROR *either the client or server is out of memory*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_get_obj_by_index(OBJ_TYPE obj_type, int obj_index, void **obj_ptr);

ct_get_obj_by_name

Given the object type and object name, this function returns the object.

Input

`obj_type` *The object type to query*

`obj_name_ptr` *Pointer to the object name*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_APPLICATION

CT_ADMINISTRATIVE_GROUP

CT_WEB_SERVER

CT_APPLICATION

CT_USER_PROPERTY_DEF

Output

`obj_ptr` *Pointer to the requested object*

Return Code

`RC_OK`	*Successfully retrieved the object*
`RC_OBJ_NOT_FOUND`	*Object not found*
`RC_INVALID_TYPE`	*Specified invalid object type*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
RC_MEMORY ERROR	*Either the client or server is out of memory*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_get_obj_by_name(OBJ_TYPE obj_type, char *obj_name_ptr, void **obj_ptr);

ct_get_objs_by_names

This function returns a range of objects based on an object type and array of object names.

Input

`obj_type`	*The object type to query*
`name_array`	*An array of names to query*
`num_of_names`	*Number of names in the array*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_APPLICATION

CT_ADMINISTRATIVE_GROUP

CT_WEB_SERVER

CT_USER_PROPERTY_DEF

Output

`array_size` *Number of array elements*

`objs_ptr` *Pointer to an array of objects*

Return Code

`RC_OK` *Successfully retrieved the objects*

`RC_OBJ_NOT_FOUND` *Objects not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_MEMORY_ERROR` *Either the client or server is out of memory*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_get_objs_by_names(OBJ_TYPE obj_type, char *name_array[], int num_of_names, int *array_size, void **obj_ptr);

ct_get_objs_by_range

This function returns a range of objects.

Input

`obj_type` *The object type to query*

`start_index` *Start index*

`end_index` *End index*

Valid Object Types

CT_USER

CT_GROUP

CT_REALM

CT_APPLICATION

CT_ADMINISTRATIVE_GROUP

CT_WEB_SERVER

CT_USER_PROPERTY_DEF

Output

`array_size`	*Number of array elements*
`objs_ptr`	*Pointer to an array of objects*

Return Code

`RC_OK`	*Successfully retrieved the objects*
`RC_OBJ_NOT_FOUND`	*Objects not found*
`RC_INVALID_TYPE`	*Specified invalid object type*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
`RC_MEMORY_ERROR`	*Either the client or server is out of memory*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_get_objs_by_range(OBJ_TYPE obj_type, int start_index, int end_index, int *array_size, void **obj_ptr);

ct_save_obj

This function saves an object.

Input

obj_type	*The object type to be saved*
obj_ptr	*Pointer to the object to save*

Valid Object Types

CT_GROUP

CT_REALM

CT_USER

CT_APPLICATION

CT_WEB_SERVER

CT_EXPLICIT_ENTITLEMENT

CT_APPLICATION_FUNCTION

CT_USER_PROPERTY_DEF

CT_USER_PROPERTY

Outputnone

Return Code

RC_OK	*Successfully saved the object*
RC_OBJ_NOT_FOUND	*Object not found*
RC_INVALID_TYPE	*Specified invalid object type*
RC_NOT_AUTHORIZED	*User not authorized to perform the function*
RC_MEMORY_ERROR	Either the client or the server is out of memory
RC_TRANSPORT_ERROR	*Error attempting to communicate with the server*

Syntax

int ct_save_obj(OBJ_TYPE obj_type, void *obj_ptr);

Relationships and Associations

The ClearTrust SecureControl API functions in this section use relationships and associations. The following list describes the types of relationships and associations available via the ClearTrust SecureControl API:

`ADMINGROUP_ADMINS`	*Administrators contained in an AdminGroup*
`ADMINGROUP_USERS`	*Users contained in an AdminGroup*
`ADMINGROUP_GROUPS`	*Groups contained in an AdminGroup*
`ADMINGROUP_REALMS`	*Realms contained in an AdminGroup*
`ADMINGROUP_WEBAPPS`	*Applications contained in an AdminGroup*
`ADMINGROUP_WEBSRVRS`	*WebServers contained in an AdminGroup*
`APP_APPFUNC`	*An Application's Application Function*
`APP_EXP_ENTITLEMENTS`	*An Application's Basic Entitlements*
`GROUP_USERS`	*Users contained in a Group*
`GROUP_REALMS`	*Realms that contain a Group*
`GROUP_EXP_ENTITLEMENTS`	*A Group's Basic Entitlements*
`REALM_GROUPS`	*Groups contained in a Realm*
`REALM_EXP_ENTITLEMENTS`	*A Realm's Basic Entitlements*
`USER_GROUPS`	*Groups that contain a User*
`USER_USERPROPS`	*UserProperties for a User*
`USER_EXP_ENTITLEMENTS`	*A User's Basic Entitlements*
`WEBSERVER_WEBURLS`	*WebURLs associated with a WebServer*

Relational Functions

This section describes the ClearTrust SecureControl relational functions.

Given the object and the object relation type, this function returns the number of objects the given object is related to for the given type.

Input

obj_rel_type	*The object relation type*
obj_ptr	*Pointer to the object to query*

Valid Relationship Types

CT_ADMINGROUP_ADMINS

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_APP_APPFUNC

CT_GROUP_USERS

CT_GROUP_REALMS

CT_GROUP_EXP_ENTITLEMENTS

CT_REALM_GROUPS

CT_REALM_EXP_ENTITLEMENTS

CT_USER_GROUPS

CT_USER_USERPROPS

CT_USER_EXP_ENTITLEMENTS

CT_APP_EXP_ENTITLEMENTS

CT_APP_WEBURLS

CT_WEBSERVER_WEBURLS

Output

`num_of_rels`	*Contains the number of objects for the specified relation type*

Return Code

`RC_OK`	*Successfully retrieved the number of objects*
`RC_INVALID_TYPE`	*Specified invalid relation type*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_get_num_of_obj_rels(OBJ_REL_TYPE obj_rel_type, void *obj_ptr, int *num_of_rels);

CT_GET_OBJ_BY_REL_INDEX

Given the object relationship type, object pointer, and the relationship index, this function returns the requested object.

Input

`obj_rel_type`	*The object relationship type to query*
`from_obj_ptr`	*The "from" object*
`rel_index`	*The index into the relationships*

Valid Relationship Types

CT_ADMINGROUP_ADMINS

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_APP_APPFUNC

CT_GROUP_USERS

CT_GROUP_REALMS

CT_GROUP_EXP_ENTITLEMENTS

CT_REALM_GROUPS

CT_REALM_EXP_ENTITLEMENTS

CT_USER_GROUPS

CT_USER_USERPROPS

CT_USER_EXP_ENTITLEMENTS

CT_APP_EXP_ENTITLEMENTS

CT_APP_WEBURLS

CT_WEBSERVER_WEBURLS

Output

`obj_ptr` *Pointer to the requested object*

Return Code

`RC_OK` *Successfully retrieved the object*

`RC_OBJ_NOT_FOUND` *Object not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_get_obj_by_rel_index(OBJ_REL_TYPE obj_rel_type, void *from_obj_ptr, int rel_index, void **obj_ptr);

CT_GET_OBJ_BY_REL_RANGE

Given the object relationship type, object ptr, and the relationship index, this function returns the requested object.

Input

obj_rel_type	*The object relationship type to query*
from_obj_ptr	*The "from" object*
rel_start	*The index to start from*
rel_end	*The index to end*

Valid Relationship Types

CT_ADMINGROUP_ADMINS

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_APP_APPFUNC

CT_GROUP_USERS

CT_GROUP_REALMS

CT_GROUP_EXP_ENTITLEMENTS

CT_REALM_GROUPS

CT_REALM_EXP_ENTITLEMENTS

CT_USER_GROUPS

CT_USER_USERPROPS

CT_USER_EXP_ENTITLEMENTS

CT_APP_EXP_ENTITLEMENTS

CT_APP_WEBURLS

CT_WEBSERVER_WEBURLS

Output

`array_size`	*Number of objects is the returned array*
`obj_ptr`	*Pointer to the returned array of objects*

Return Code

`RC_OK`	*Successfully retrieved the object*
`RC_OBJ_NOT_FOUND`	*Object not found*
`RC_INVALID_TYPE`	*Specified invalid object type*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*
`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_get_obj_by_rel_range(OBJ_REL_TYPE obj_rel_type, void *from_obj_ptr, int rel_start, int rel_end, int *array_size, void **obj_ptr);

CT_GET_OBJ_BY_REL_NAME

Given the object relationship type, object ptr, and the name of the related object, this function returns the related object.

Input

obj_rel_type *The object relationship type to query*

from_obj_ptr *The "from" object*

rel_obj_name *The name of the related object*

Valid Relationship Types

CT_ADMINGROUP_ADMINS

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_GROUP_USERS

CT_GROUP_REALMS

CT_REALM_GROUPS

CT_USER_GROUPS

CT_USER_USERPROPS

Output

`obj_ptr`	*Pointer to the requested object*

Return Code

`RC_OK`	*Successfully retrieved the object*
`RC_OBJ_NOT_FOUND`	*Object not found*
`RC_INVALID_TYPE`	*Specified invalid object type*
`RC_NOT_AUTHORIZED`	*User not authorized to perform the function*

`RC_TRANSPORT_ERROR`	*Error attempting to communicate with the server*

Syntax

int ct_get_obj_by_rel_name(OBJ_REL_TYPE obj_rel_type, void *from_obj_ptr, char *rel_obj_name, void **obj_ptr);

CT_GET_OBJS_BY_REL_NAMES

This function returns an array of objects based on an array of object names.

Input

`obj_rel_type`	*The object relationship type to query*
`from_obj_ptr`	*The "from" object*
`name_array`	*Array of names*
`num_of_names`	*Number of names in the array*

Valid Relationship Types

CT_ADMINGROUP_ADMINS

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_GROUP_USERS

CT_GROUP_REALMS

CT_REALM_GROUPS

CT_USER_GROUPS

CT_USER_USERPROPS

Output

`array_size` *Number of objects is the returned array*

`obj_ptr` *Pointer to the returned array of objects*

Return Code

`RC_OK` *Successfully retrieved the object*

`RC_OBJ_NOT_FOUND` *Object not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_get_objs_by_rel_names(OBJ_REL_TYPE obj_rel_type, void *from_obj_ptr, char *name_array[], int num_of_names, int *array_size, void **obj_ptr);

CT_SET_REL

This function sets a relationship for the specified relationship type between the two specified objects.

Note: *Some object's relationships are fixed at the time of the object's creation. Thus,* ***`ct_set_rel`*** *only applies to a subset of relationships.*

Input

`obj_rel_type` *The object relationship type to set*

`from_obj_ptr` *The "from" object*

`to_obj_ptr` *The "to" object*

Valid Relationship Types

CT_ADMINGROUP_USERS

CT_ADMINGROUP_GROUPS

CT_ADMINGROUP_REALMS

CT_ADMINGROUP_WEBAPPS

CT_ADMINGROUP_WEBSRVRS

CT_GROUP_USERS

CT_GROUP_REALMS

CT_REALM_GROUPS

CT_USER_GROUPS

Outputnone

Return Code

`RC_OK` *Successfully set the relationship*

`RC_OBJ_NOT_FOUND` *Object not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_set_rel(OBJ_REL_TYPE obj_rel_type, void *from_obj_ptr, void *to_obj_ptr);

CT_DEL_REL

The function **`ct_del_rel`** deletes a relationship for the specified relationship type between the two specified objects.

Input

`obj_rel_type` *The object relationship type to delete*

`from_obj_ptr` *The "from" object*

`to_obj_ptr` *The "to" object*

Valid Relationship Types

CT_GROUP_USERS

CT_GROUP_REALMS

CT_REALM_GROUPS

CT_USER_GROUPS

CT_USER_ADMINS

Outputnone

Return Code

`RC_OK` *Successfully set the relationship*

`RC_OBJ_NOT_FOUND` *Object not found*

`RC_INVALID_TYPE` *Specified invalid object type*

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_del_rel(int obj_rel_type, void *from_obj_ptr, void *to_obj_ptr);

Runtime Functions

You use runtime functions to consult or check the ClearTrust SecureControl database. This section describes runtime commands.

ct_check_access

This function checks a User's accessibility to a URI. It only returns TRUE for a valid user, as in ct_validate_user.

Note: *You do not need to supply an Administrative name, password, or Role on the connect statement to use this function.*

Input

user_name *Pointer to a User's name*

password *Pointer to a User's password. If the password is Null, no password checking is done.*

web_server_name*Pointer to the URI's WebServer name*

uri *Pointer to the URI*

Output

is_accessible*Pointer to a Boolean value:*

TRUE - URI is accessible

FALSE - URI isn't accessible

Return Code

RC_OK *Access successfully determined*

RC_OBJ_NOT_FOUND*User not found*

RC_NO_AUTH_SERVERS*No Authorization Servers were available to process request*

RC_TRANSPORT_ERROR*Error attempting to communicate with the server*

Syntax

int ct_check_access(char *user_name, char *password, char *web_server_name, char *uri, CT_BOOLEAN *is_accessible);

ct_check_function

This function checks a User's accessibility to an Application's function. Only returns TRUE for a valid user, as in ct_validate_user.

Input

user_name *Pointer to a User's name or Distinguished Name*

password *Pointer to a User's password. If the password is Null, no password checking is done.*

app_name *Pointer to the name of the Application*

func_name *Pointer to the name of the Application's function*

Output

is_accessible *Pointer to a Boolean value:*

TRUE - User can access the Application's function

FALSE – User is denied access to the Application's function

Return Code

RC_OK *Access successfully determined*

RC_OBJ_NOT_FOUND*User not found*

RC_NO_AUTH_SERVERS*No Authorization Servers were available to process request.*

RC_TRANSPORT_ERROR*Error attempting to communicate with the server.*

Syntax

int ct_check_function(char *user_name, char *password, char *app_name, char *func_name, CT_BOOLEAN *is_accessible);

ctxt_check_password (and ct_check_password)

This function only checks a User's password. It does not check account validation.

Note: *You do not need to supply an Administrative name, password, or Role on the connect statement to use this function.*

Input

context

user_name *Pointer to a User's name or Distinguished Name*

password *Pointer to a User's password.*

Output

is_valid *Pointer to a Boolean value*

TRUE - User is found and password is valid

FALSE - User is found and password is invalid

Return Code

RC_OK *Access was determined*

RC_OBJ_NOT_FOUND *User not found*

RC_NOT_AUTHORIZED*User not authorized to perform the function*

RC_NO_AUTH_SERVERS*No Authorization Servers were available to process request.*

RC_TRANSPORT_ERROR*Error attempting to communicate with the server*

Syntax

int ct_check_password(char *user_name, char *password, CT_BOOLEAN *is_valid);

ctxt_create_app_func (and ct_create_app_func)

This function creates an Application Function associated with an Application. An Application Function represents a protected set of functionality within an Application. This set of functionality can be protected through the creation of a Basic Entitlement.

Input

context *token for CT_Context*

app_ptr *Pointer to the Application Function's Application*

app_func_ptr *Pointer to the Application Function*

Output

app_func_ptr *The Application Function's id will be set*

Return Code

RC_OK *Successfully created an Application Function*

RC_OBJ_NOT_FOUND*Object not found*

RC_NOT_AUTHORIZED*User not authorized to perform the function*

RC_TRANSPORT_ERROR*Error attempting to communicate with the server*

Syntax

```
int ctxt_create_app_func(CT_Context context, CT_Application
*app_ptr, CT_ApplicationFunction *app_func_ptr);


int
ct_create_app_func(CT_Application,*app_ptr,CT_ApplicationFun
ction *app_func_ptr);
```

ctxt_create_app_url (and ct_create_app_url)

Creates an Application URL associated with an Application and WebServer. An Application URL represents a resource labeled by a URI and associated with a particular Application and Web Server.

Input

app_ptr Ptr to the Application Url's Application

webserver_ptr ptr to the Application Url's Webserver

app_url_ptr ptr to Application Url

Output

app_url_ptr The Application Url's id, app_id, and websrvr_id will be set

Return code

RC_OK Successfully created an Application URL

RC_OBJ_NOT_FOUND - The Application or the WebServer can not be found.

RC_DUPLICATE_OBJ_ERROR - An ApplicationURL already exists in the system.

RC_NOT_AUTHORIZED - User not authorized to perform the function

RC_TRANSPORT_ERROR - Error attempting to communicate with the server

Syntax

```
int ct_create_app_url(CT_Application *app_ptr, CT_WebServer
*webserver_ptr, CT_ApplicationURL *app_url_ptr);


int ctxt_create_app_url(CT_Context context, CT_Application
*app_ptr, CT_WebServer *webserver_ptr, CT_ApplicationURL
*app_url_ptr);
```

ct_get_apps_for_user

This function retrieves the corresponding applications for which a user has 'ALLOW' privileges for an Application Function.

Input

user_name	*Pointer to a User's name*
func_name	*Pointer to the name of the Application's Function*

Output

array_size	*Number of array elements*
apps_ptr	*Pointer to an array of Applications*

Return Code

RC_OK	*Successfully retrieved the objects*
RC_OBJ_NOT_FOUND	*Objects not found*
RC_INVALID_TYPE	Specified invalid object type

`RC_NOT_AUTHORIZED` *User not authorized to perform the function*

`RC_MEMORY_ERROR` *Either the client or server is out of memory*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

int ct_get_apps_for_user (char* user_name, char* func_name, int *array_size, CT_Application **apps_ptr)

ctxt_validate_user (and ct_validate_user)

This function validates a User's account. The user can have an invalid account for any of the following reasons:

- Invalid username
- Incorrect password for given user
- User's password is locked
- User's password is expired
- User's Start Date is after current Authorization Server System Time
- User's End Date is before current Authorization Server System Time

NOTE: All times are internally computed as GMT to preserve functionality.

Input

user_name Pointer to a User's name or Distinguished Name

password Pointer to a User's password

NOTE: You do not need to supply an Administrative name, password, or Role on the connect statement to use this function.

Output

`is_valid` *Pointer to a Boolean value:*

TRUE - User is found and password validity was determined

FALSE - User is found and password is invalid

Return Code

`RC_OK` *User is found*

`RC_OBJ_NOT_FOUND` *User not found*

`RC_NO_AUTH_SERVERS` *No Authorization Servers were available to process request.*

`RC_TRANSPORT_ERROR` *Error attempting to communicate with the server*

Syntax

```
int ctxt_validate_user(CT_Context context, char *user_name,
char *password, CT_BOOLEAN *is_valid);


int ct_validate_user(char *user_name, char *password, BOOLEAN
*is_valid);
```

ClearTrust SecureControl API Sample Code and Examples

C Sample Code

You can find a sample C application in `driver.c` when you install the SecureControl servers. This application connects to a ClearTrust SecureControl server, sets the Application's Administrator, creates a single User, creates a single Group, puts the User in the Group, and disconnects from the ClearTrust SecureControl server.

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


#include "ct_commands.h"
/*
*
*This driver is an example of how to use the SecureControl-
API.  Note *there are a few prerequistes before successfully
using the API: The *SecureControl Database has to be initially
defined with the following *resources:
```

```
*
*  1)  There has to be at least one superuser defined.  This is
*      required for API connection
*  2)  There has to be Administrative Roles defined.  This is
*      required for API connection
*  3)  All CT_UserPropertyDefinitions have to be pre-defined.
They
*      cannot be defined through the API.
*  4)  All WebServers have to be pre-defined.  They cannot be
*      defined through the API
*/
```

Java Examples

To use the Java API, you need to add the following files to the classpath:

```
<SecCtrlPath>\Installs\API\Java\Classes\SCAPI.jar
<SecCtrlPath>\Classes\jg13_1_0.jar
```

To run the APIDriver example from `<SecCtrlPath>\Installs\API\Java\Examples`, you need to add '**.**' or `<SecCtrlPath>\Installs\API\Java\Examples` to the classpath.

NOTE: For information on the Java API, refer to the Javadoc provided with the ClearTrust SecureControl installation. The Javadoc is located in the `api\java\doc` directory on the installation disk.

Linking with the ClearTrust SecureControl C API

The ClearTrust SecureControl C API is a statically linked library.

Library for Unix

Our current C-API has been tested on two platforms, the Solaris 2.5.1 platform and the Windows NT v4.0 SP4 platform. For these platforms, we have employed the following compilers and their corresponding versions:

Solaris platform: gcc version 2.95.1 (freely available from website gcc.gnu.org)

Windows NT/95 platform: msdev studio 5.0 (97)

The ClearTrust SecureControl C API library for Unix is `libctapi.a`. You must link your code with the following options:

```
gcc -o -lctapi -lsocket -lnsl -lssl -lcrypto
```

Library for Windows NT

The ClearTrust SecureControl C API library is thread safe and uses the NT multi-threading library; consequently, your code also must be thread safe and use the multi-threading library.

Compile Options

```
_MT
```

Linker Options (additional libraries)

```
ctapi.lib libcmt.lib wsock32.lib libeay32.lib ssleay32.lib
```

You must also override the default libraries.

Chapter 3
The ClearTrust Web Server Plug-in API

The ClearTrust SecureControl WebServer Plug-in provides an API that allows developers to extend and customize the functionality of the ClearTrust SecureControl Plug-in. Unlike the ClearTrust SecureControl API, the ClearTrust SecureControl Plug-in API does not modify the ClearTrust SecureControl database, rather it controls the behavior of the ClearTrust SecureControl Plug-in during the authentication and authorization processing.

For example, you could extend the functionality of the ClearTrust SecureControl WebServer Plug-in in the following ways:

- Create an extension directing the Web Server to a custom HTML file for different return codes from the ClearTrust SecureControl Authorization Server.
- Create an extension providing custom logging.
- Create an extension providing custom authentication of Users.
- Create an extension integrating the ClearTrust SecureControl Plug-in with propriety WebServer Plug-ins or third party Plug-ins.

This chapter provides an overview of the ClearTrust Plug-in API, including processing flows and logic loops, and provides an overview about how to integrate your own custom extension—the ClearTrust Plug-in Extension (PIX) that you write—into your Web server environment. It includes the following sections:

- Extending Functions of the ClearTrust Web Server Plug-in
- Compiling and Linking with the ClearTrust SecureControl Plug-in API
- Custom Authentication Example

- Custom Error Pages Example

Extending Functions of the ClearTrust Web Server Plug-in

ClearTrust SecureControl Plug-in Extensions are implemented using a call-back scheme. Much like the call-back mechanisms in the Netscape and IIS servers, a ClearTrust SecureControl Plug-in Extension must register itself to the ClearTrust SecureControl Plug-in and define the various routines to call when processing a URL request. Figure 1-1 illustrates the relationship among a ClearTrust SecureControl Plug-in Extension, the ClearTrust SecureControl Plug-in, various Web Servers, and their respective APIs.

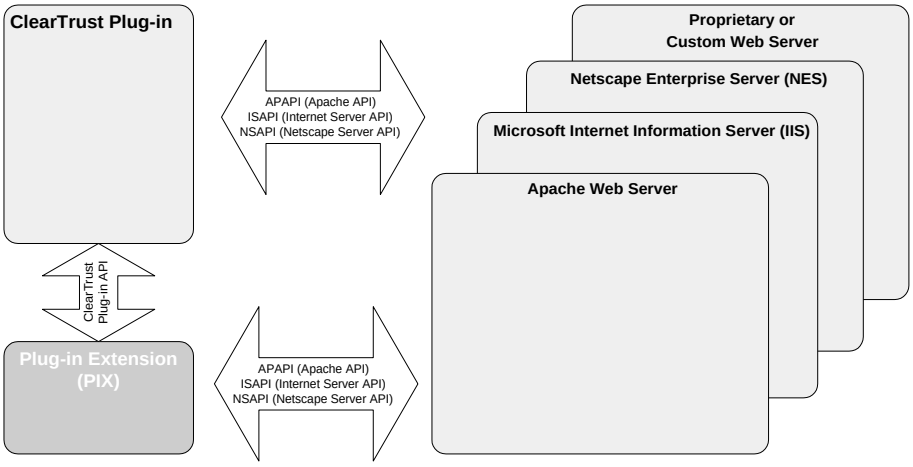


FIGURE 1-1: The ClearTrust SecureControl Plug-in API can be used by customers and third-parties to develop custom code modules, called Plug-in Extensions, or PIX, that extend the functionality of the ClearTrust Plug-in. Calls made using the Plug-in API are available to the plug-in to which it's registered and to the Web Server.

As shown in Figure 1-1, ClearTrust SecureControl Web Server Plug-ins are implemented using the APIs of the respective Web Server vendors, specifically, Apache, Microsoft, or Netscape. Plug-in Extensions (PIX) are code modules that control the behavior of the ClearTrust SecureControl Web Server Plug-in during authentication and authorization processing. For example, you can extend the functionality of the ClearTrust SecureControl Web Server Plug-in to:

- Direct the Web Server to a custom HTML file for different return codes from the ClearTrust SecureControl Authorization Server.
- Provide custom logging.
- Provide custom authentication of users.
- Integrate the ClearTrust SecureControl Plug-in with proprietary Web Server plug-ins or third party plug-ins.

Integrating a custom extension involves several steps, from developing the code for the actual authentication implementation (or implementing a third-party vendor's API in your code) to integrating the object code into your runtime environment by modifying the ClearTrust SecureControl Web Server Plug-in's configuration files. Developers use ClearTrust Plug-in API function calls in their code to create new Plug-in Extensions (PIX); the code must model one of the Web Server Plug-in's five processing phase handlers.

Before discussing the specific implementation details, here's a brief overview of the default ClearTrust Web Server Plug-in five-phase request handler process.

How the ClearTrust Plug-in Processes a URI Request

During a URI request, the Web Server invokes the ClearTrust SecureControl Plug-in to perform authentication and authorization. The ClearTrust SecureControl Plug-in processes the request by executing a sequence of *phases*. During each phase, the ClearTrust SecureControl Plug-in 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 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 1-2).

As the ClearTrust Web Server Plug-in 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 Plug-in skips the default handler.
- If it returns FALSE, the default handler is invoked.

After the phase handlers have completed, the status handler is invoked to handle the result from the phase handler. The Plug-in 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, the default status handler is still invoked. When the default status handler is invoked, only logging and messaging actions are allowed; processing for the URI request ceases.
- If the custom status handler returns FALSE, the default handler is called and invokes a Web Server function and/or executes the next phase handler.

Figure 3-1 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.

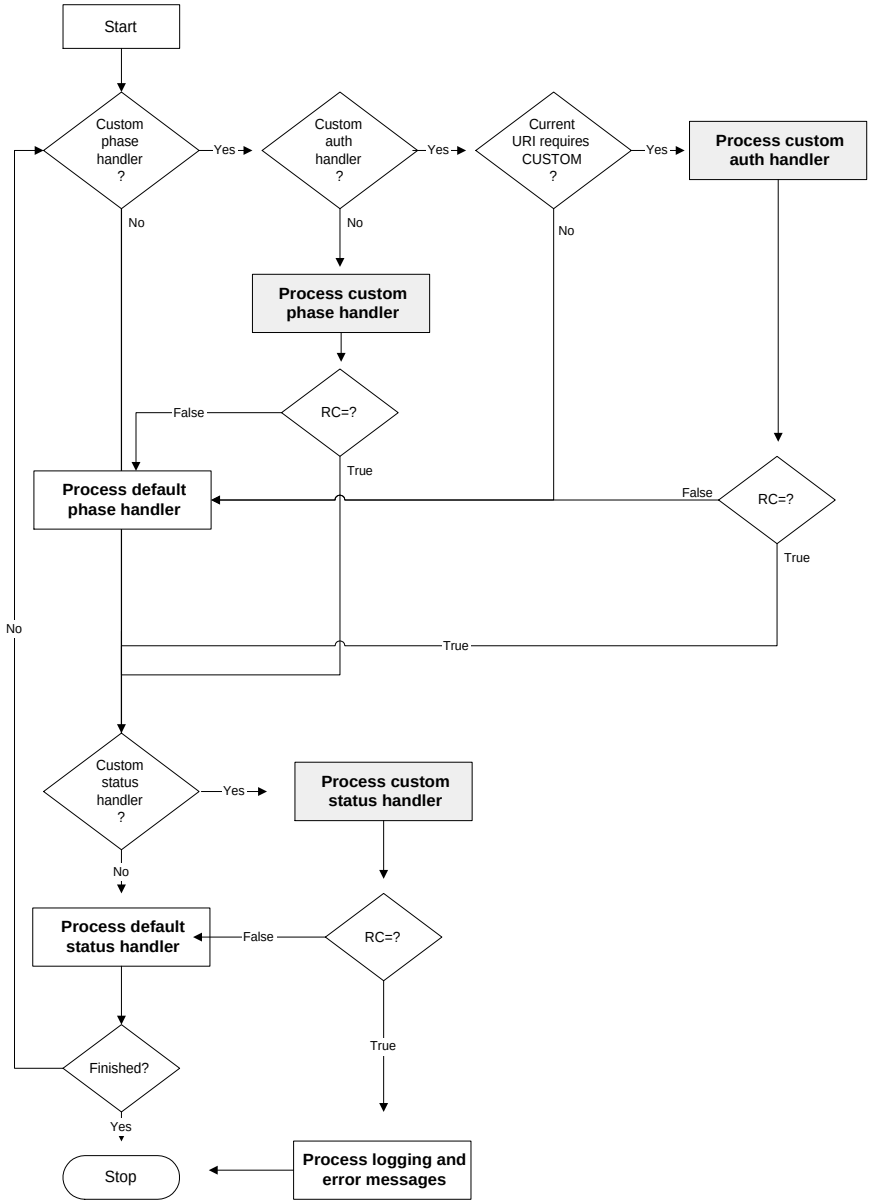


Figure 3-1. SecureControl Plug-in Processing Loop

Phase Handlers Overview

During the processing of a URI request, the ClearTrust SecureControl Web Server Plug-in executes a sequence of five phases to perform authentication, authorization, and single sign-on, ultimately determining the accessibility of the URI. The five phase handlers are:

- Path Check Handler
- Session Handler
- Authentication Handler
- Authorization Handler
- Cookie Handler

PATH CHECK HANDLER

The Path Check Handler determines whether the request URI is protected. The handler invokes the ClearTrust SecureControl 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 executes the Session phase. In addition, the Path Check Handler sets the allowable authentication modes for the URI as configured in the Plug-in `Default.conf` file (securecontrol.plugin.auth_resource_list) in the request table. The authentication mode key is `CT_ALLOWABLE_AUTH_MODES`; you can set multiple modes.

SESSION HANDLER

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 default status handler executes the Authentication phase.

AUTHENTICATION HANDLER

The Authentication Handler authenticates the User based on the allowable authentication modes (CT_ALLOWABLE_AUTH_MODES) set by the Path Check Handler.

First, the Authentication Handler gets the allowable authentication modes required by the URI from the request table (CT_ALLOWABLE_AUTH_MODES). Using this list, the Authentication Handler then checks to see if the User has already authenticated with any of the allowable authentication modes.

- If the User has already authenticated with any of the required authentication modes, the Authentication Handler sets a status code of CT_ACCESS_REQUIRED.
- If the User has not authenticated with any of the required authentication modes, the Authentication Handler authenticates the User with the first authentication mode on the 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 the authentication mode is "BASIC" during non-form-based authentication, the authentication is deferred to the Authentication Handler.
- 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 against the appropriate authentication mode.

Based on what is returned from the authentication mode, the Authentication Handler sets the status with 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 mode and therefore will not need to authenticate against this mode in the future. The Authentication Handler also sets the authentication mode (CT_AUTH_MODE) in the request table. The Authentication Handler uses CT_AUTH_MODE to determine whether or not it needs to authenticate the User.
- If the authentication is not successful, the status code is handled as follows:

- The Web Server is instructed to return a `WWW-Authenticate` response (`HTTP 401`) or is redirected to a custom logon page for the following return codes:

  `CT_AUTH_BAD_USERNAME` The User is not defined in the ClearTrust SecureControl 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_USER_LOCKED_OUT` The 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 inspects the `CT_AUTH_MODE` in the request table. If `CT_AUTH_MODE` is set to `UPW` or `UDN`, the Authorization Handler instructs the ClearTrust SecureControl Authorization Server to perform authentication and authorization in a single step. Generally, the Authorization performs only authorization but the Authorization Handler may set the status code with values defined for both the Authentication and the Authorization phase.

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 ClearTrust SecureControl Plug-in provides users with a 1k 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 a ClearTrust SecureControl cookie. For example, you may want to create a Plug-in extension 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 1-2 illustrates how the five phase handlers process a URI request.

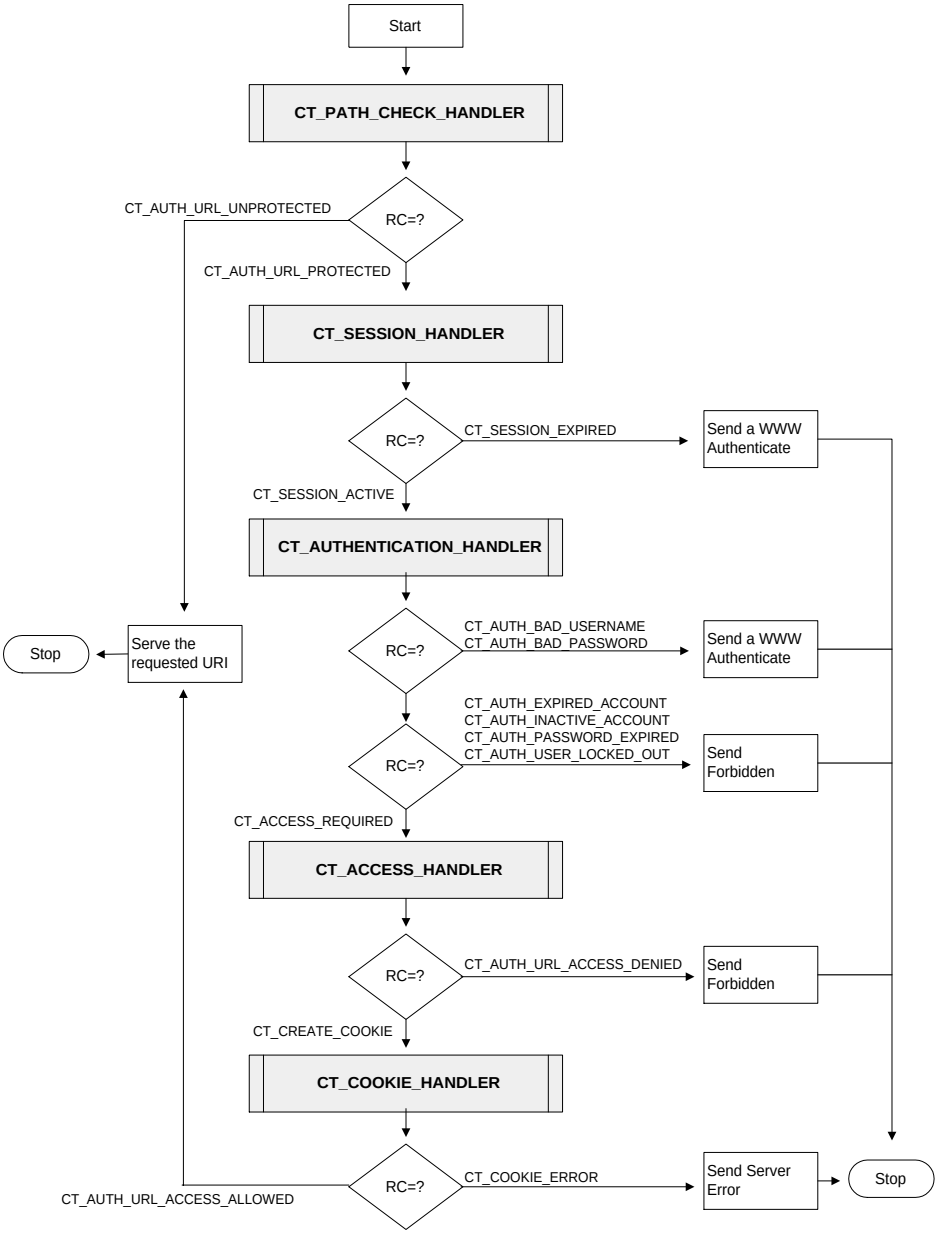


TABLE 1-2: Phase Processing a URI Request

Implementation Details: Custom Phase Handlers

If you wish to implement an authentication mechanism that isn't currently supported as part of the core ClearTrust product or your Web Server—a proprietary mechanism, or a PKI (private key infrastructure) implementation, for example—you can extend the functionality of the ClearTrust Web Server Plug-in by using the Plug-in API to write your own phase handler, which can then be called during URI request processing. To integrate the custom phase handler (the PIX) into the ClearTrust Web Server Plug-in processing loop, your code must call the appropriate handler key (see Table 1-1).

The ClearTrust SecureControl Plug-in API is installed as part of the ClearTrust Web Server Plug-in installation. You'll find the header files in the <securant_install_dir>/plugins/include subdirectory. The Plug-in API comprises three header files and the libary file that's specific to the Web Server Plug-in that you'll be extending:

```
ct_table.h

ct_function_table.h

ct_request_data.h

ct-iis40-plugin.lib (for Microsoft IIS/Windows NT)

ct-nscp350nt-plugin.lib (Netscape Enterprise Server/Windows
NT)

ct-nscp201sol-plugin.so (Netscape Enterprise Server/Solaris)

libsc42-apache13x-plugin.a (Apache/Solaris)
```

See the "ClearTrust SecureControl Plug-in API Reference" later in this section for more information. Presuming that you've installed the ClearTrust Web Server Plug-in and have it integrated with your Web Server (Apache, Microsoft IIS, or Netscape Enterprise Server), integrating a custom authentication mechanism in your ClearTrust SecureControl implementation involves only a few steps:

1. Developing the phase handler code, which will include calls to the third-party security server using the third-party vendor-provided API and calls to the ClearTrust Web Server Plug-in using the Plug-in API.
2. Compiling, testing, and debugging your custom phase handler.

3 Modifying the ClearTrust Web Server Plug-in's configuration file (<securant_install_dir>/plugins/conf/default.conf) to use the word "CUSTOM" as the authentication mechanism.

Note that your specific ClearTrust SecureControl configuration can include multiple PIX custom phase handlers, in addition to the default phase handlers.

Integrating Custom Handlers

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 Plug-in. To customize the action that results from a phase handler or to alter the flow of URI request processing, you must integrate your custom code by registering your PIX in this function table. You do this by first obtaining the table (using the `ct_get_function_table function call)` and then adding your function to the table using the `ct_table_put` function call. For example, your code could include something like this:

```
ct_table_ptr ct_func_table = ct_get_function_table();
ct_table_put(ct_func_table, CT_AUTHENTICATION_HANDLER, my-cust-auth);
```

The function table structure and definitions are found in the `ct_function_table.h` header file, which is contained in the Securant\plugin installation directory when you install the Web Server Plug-in. the Addressability for the function table is obtained through the function call . Once you have the table, you can add your new functions with the `function` call. The following table describes the various keys.

TABLE 4-1: Handler Keys and their Descriptions

Key	Description
`CT_STATUS_HANDLER`	The status handler. The status handler is called after the execution of the phase handlers. Developers should reference this key to customize actions that result from a phase handler or to alter the URI request processing flow.
`CT_SESSION_HANDLER`	The session handler. The session handler determines determining whether the Single Sign-On session has expired.

Key	Description
CT_PATH_CHECK_HANDLER	The path check handler. The path check handler determines whether or not the requested URI is protected.
CT_AUTHENTICATION_HANDLER	The authentication handler. The authentication handler determines whether or not the user is valid.
CT_ACCESS_HANDLER	The access handler. The access handler determines whether or not the user has access to requested URI.
CT_COOKIE_HANDLER	The cookie handler. The cookie handler creates the cookie used in the Single Sign-On processing.

Compiling and Linking with the ClearTrust SecureControl Plug-in API

This section provides guidelines for compiling and linking with the ClearTrust SecureControl Plug-in API on the following browsers/platforms:

- Microsoft IIS
- Netscape on Windows NT
- Netscape on Solaris

Compiling and Linking for Microsoft IIS

This section tells you how to compile and link with the ClearTrust SecureControl Plug-in API for Microsoft IIS.

Compiling for IIS

Compile Options:WIN32,_DEBUG,_WINDOWS,MSIIS,WINDOWS

Additional Include Libraries: `<plugin_install_dir>\include`

Linking for IIS

Additional Libraries: `ct-iis40-plugin.lib`

Additional Library Path:`<plugin_install_dir>\lib`

Additionally, IIS filter requires a `.def` file containing external declaration.

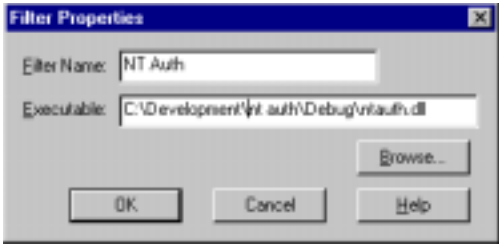
The following is an example of a `.def` file:

```
LIBRARY NTAUTH
DESCRIPTION 'NT Native Authentication Filter'
```

```
EXPORTS
   DllMain
   GetFilterVersion
   HttpFilterProc
```

Installing for IIS

To install your PIX on IIS, you must add it to the IIS properties in the Microsoft Manager Console at the site level. Here is an example of adding the filter properties.



Compiling and Linking for Netscape (Windows NT)

This section tells you how to compile and link with the ClearTrust SecureControl Plug-in API for Netscape on Windows NT.

Compiling for Netscape/NT

Compile Options:
WIN32,_DEBUG,_WINDOWS,NETSCAPE,WINDOWS,XP_WIN32

Additional Include Libraries: `<plugin_install_dir>\include`

and

<netscape_server_dir>\include

Linking for Netscape/NT

Additional Libraries: `ct-nscp300nt-plugin.lib`, `ct-nsft300nt-plugin.lib`, or `ct-nscp350nt-plugin.lib`

Additional Library Path: `<plugin_install_dir>\lib`

Installing for Netscape/NT

In order to install the ClearTrust SecureControl Plug-in on Netscape/NT, you must modify the `obj.conf` file. The installation requires the **`Init fn=load-modules`** directive defining the name of the redirect dll. The second **`Init fn=nt-auth-init`** directive defines the initialization routine of the redirect dll. The following is an example of obj.conf:

```
Init fn="load-modules" funcs="nt-auth-init"
shlib="C:/development/nt_auth/debug/ntauth.dll"
```

```
Init fn="nt-auth-init"
```

For more information on installing and configuring a Netscape Plug-in, refer to your Netscape documentation.

Compiling and Linking for Netscape/UNIX

This section tells you how to compile and link with the ClearTrust SecureControl Plug-in API for Netscape/UNIX.

Compiling for Netscape/UNIX

Compile Options: `-DNETSCAPE -DFILE_UNIX -DXP_UNIX`

Additional Include Libraries: `<plugin_install_dir>/include`

and

<netscape_server_dir>\include

Linking for Netscape/UNIX

Additional libraries: `/libct-nscp201sol-plugin.so`

Additional Library Path:`<plugin_install_dir>/lib`

Installing for Netscape/UNIX

In order to install the ClearTrust SecureControl Plug-in on Netscape/UNIX, you must modify the `obj.conf` file. The installation requires the **`Init fn=load-modules`** directive defining the name of the redirect dll. The second **`Init fn=nt-auth-init`** directive defines the initialization routine of the redirect dll. The following is an example `obj.conf`:

```
Init fn="load-modules" funcs="nt-auth-init"
shlib="/opt/cairo/dev/auth/nt_auth/ ntauth.so"
```

```
Init fn="nt-auth-init"
```

For more information on installing and configuring a Netscape Plug-in, refer to your Netscape documentation.

Custom Authentication Example

You can use the ClearTrust SecureControl Plug-in API to perform custom authentication. The following nt_auth.c ClearTrust SecureControl Plug-in Extension shows how to replace the ClearTrust SecureControl authentication with native NT authentication. In order for this to work, the userIDs of the registered users in ClearTrust SecureControl must be identical to their NT userIDs.

Native NT Authentication

During Web Server initialization, the ClearTrust SecureControl Plug-in Extension registers its authentication handler in the function table. When the authentication handler is driven, the extension checks whether or not a user and password has been set. If they have been set, the ClearTrust SecureControl Plug-in Extension performs NT native authentication, sets the status, and returns a TRUE, which indicates that it has handled the authentication phase.

```
/*

nt_auth.c

Copyright (c) 1998 Securant Technologies, Inc.

All rights reserved.

This module demonstrates how you can extend the functionality
of the ClearTrust Plug-in.

This sample works with IIS and Netscape/NT. To compile for
IIS, set the MSIIS compiler directive, for Netscape set the
NETSCAPE compiler directive.

nt_auth replaces the ClearTrust authentication with NT native
authentication. To extend the ClearTrust Plug-in
functionality, you must:

1. During initialization of the Plug-in, register the phase
handler in the function table. This module registers the
nt_authenticate routine for the AUTHENTICATION phase handler
in the routine:
register_auth_handler
```

2. During the execution of the handler, set the status and return a TRUE if the handler handled this phase (thus skipping the default handler) or return a FALSE if the handler didn’t handle the phase.

During authentication the extension checks whether or not a user and password has been set. If they have, then the Plug-in performs the NT native authentication and returns a TRUE indicating that it has handle the authentication phase.**/

```
#include <stdio.h>

// Windows header files

#include <windows.h>

#include <winnt.h>

// ClearTrust header files

#include "ct_function_table.h"

#include "ct_request_data.h"

// Prototype for registering our custom authentication routine

void register_auth_handler();

// Prototype for the nt authentication

int nt_authenticate (const ct_server_parms *server_parms, ct_table_ptr ct_req_table);

#ifdef MSIIS

/***IIS requires three functions for a successful filter:

DllMain

GetFilterVersion

HttpFilterProc ***/

BOOL

WINAPI

DllMain(

     IN HINSTANCE hinstDll,

     IN DWORD     fdwReason,

     IN LPVOID    lpvContext OPTIONAL

)

{
```

```
    switch (fdwReason)
    {
case DLL_PROCESS_ATTACH:
//
// We don't care about thread attach/detach notifications
//
DisableThreadLibraryCalls(hinstDll);
register_auth_handler();
break;
default:
break;
}
return TRUE;
}
/**** Return the filter version, required by IIS***/
BOOL
WINAPI
GetFilterVersion(
    HTTP_FILTER_VERSION * pVer
)
{
pVer->dwFilterVersion = HTTP_FILTER_REVISION;
pVer->dwFlags = (0); // No call backs for this filter
sprintf(pVer->lpszFilterDesc, "NT Native
Authenticationtfilter, Version");
    return TRUE;            // Return true to make it happen
}
/**** This routine should never be driven, but IIS requires
it there***/
DWORD
WINAPI
HttpFilterProc(
```

```
    HTTP_FILTER_CONTEXT *      pfc,
    DWORD                      NotificationType,
    VOID *                     pvData
)
{
    return TRUE;
}
#endif


/**** Netscape initialization, much simpler than IIS, isn't
it?**/
#ifdef NETSCAPE
NSAPI_PUBLIC int nt_auth_init(pblock *param, Session *sn,
Request *rq)
{
register_auth_handler();
return REQ_PROCEED;
}
#endif
/**** Routine to register our authentication function to the
ClearTrust Plug-in**/
void register_auth_handler()
{
ct_table_ptr ct_func_table = ct_get_function_table();


ct_table_put(ct_func_table, CT_AUTHENTICATION_HANDLER,
nt_authenticate);
}
/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 auth handler makes the
NT API call authenticating the user and then returns a TRUE
directing ClearTrust not to perform default authentication/

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_USER);

LPTSTR lpszPassword = ct_table_find(ct_req_table,
CT_PASSWORD);


// If the User and Password is 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 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 isAuthenicated isn't 0, then the user is authenticated

if (bIsAuthenticated) {

// Set the AUTH_MODE to UID. This tells ClearTrust
```

```
// that only the ID is valid. ClearTrust then performs
// access checking without performing authentication
ct_table_put (ct_req_table, CT_AUTH_MODE, "UID");
// Force access checking
SET_STATUS(ct_req_table, CT_CHECK_ACCESS_REQUIRED);
} 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; // We're handling the Authentication stage,
proceed directly
                  // to the status handler
}
return bHandled;
}
```

Custom Error Pages Example

You can use the ClearTrust SecureControl Plug-in API to return a custom page when a User is denied access to a ClearTrust SecureControl protected resource.

The following ClearTrust SecureControl Plug-in Extension shows how to replace the requested URI with a new URI for various error codes that ClearTrust SecureControl returns.

During Web Server initialization, the ClearTrust SecureControl Plug-in Extension 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 custom error page and sets the status to `CT_AUTH_URL_ACCESS_ALLOWED`. This forces the ClearTrust SecureControl Plug-in to serve up the new URI.

```
/*
 * redirect.c
 * Copyright (c) 1999 Securant Technologies, Inc.
 * All rights reserved.
 *
 * This module demonstrates how you can extend the
functionality of
 * the ClearTrust Plugin.
 *
 * This sample works with IIS, Netscape/NT, and Netscape/Unix.
 * To compile for IIS, set the MSIIS compiler directive, for
Netscape set
 * the NETSCAPE compiler directive
 *
 * redirect replaces the requested URI with a new URI for
various
 * error codes returned from ClearTrust. During initialization
of the
 * plugin, this module registers the status handler in the
function table.
 *
 * The status handler checks the current status, if the status
is one
 * we wish to return a custom error page, then replace the
requested URI
 * with the custom error page and set the status to
```

```
 * CT_AUTH_URL_ACCESS_ALLOWED which forces the ClearTrust
Plugin to serve
 * up the new URI.
 *
 */
#include <stdio.h>
// Windows header files
#include <windows.h>
#include <winnt.h>
// ClearTrust header files
#include "ct_function_table.h"
#include "ct_request_data.h"
// Prototype for registering our custom status handler
void register_status_handler();
// Prototype for status handler
int handle_status (const ct_server_parms *server_parms,
ct_table_ptr ct_req_table);
// Module Definitions for the customer error pages
#define REGISTER_USER_PAGE      "/reg-user.html"
#define RE_REGISTER_USER_PAGE   "/re-reg-user.html"
#define USER_FORBIDDEN_PAGE     "/forbidden.html"
#ifdef MSIIS
/**
*
*IIS requires three functions for a successful filter:
*
*DllMain
*GetFilterVersion
*HttpFilterProc
*
**/
BOOL
```

```
WINAPI
DllMain(
     IN HINSTANCE hinstDll,
     IN DWORD     fdwReason,
     IN LPVOID    lpvContext OPTIONAL
)
{
    switch (fdwReason)
    {
case DLL_PROCESS_ATTACH:
//
// We don't care about thread attach/detach notifications
//
DisableThreadLibraryCalls(hinstDll);
register_status_handler();
break;
default:
break;
}
return TRUE;
}
/**
*
* Return the filter version, required by IIS
*
**/
BOOL
WINAPI
GetFilterVersion(
    HTTP_FILTER_VERSION * pVer
)
{
```

```
pVer->dwFilterVersion = HTTP_FILTER_REVISION;
pVer->dwFlags = (0); // No call backs for this filter
sprintf(pVer->lpszFilterDesc, "Custom error handler");
    return TRUE;            // Return true to make it happen
}
/**
*
* This routine should never be driven, but IIS requires it
there
*
**/
DWORD
WINAPI
HttpFilterProc(
    HTTP_FILTER_CONTEXT *      pfc,
    DWORD                      NotificationType,
    VOID *                     pvData
)
{
    return TRUE;
}
#endif
/**
*
* Netscape initialization
*
*/
#ifdef NETSCAPE
NSAPI_PUBLIC int redirect_init(pblock *param, Session *sn,
Request *rq)
{
register_status_handler();
```

```
    return REQ_PROCEED;
}
#endif
/**
*
* Routine to register our status handler to the ClearTrust
Plugin
*
*/
void register_status_handler()
{
ct_table_ptr ct_func_table = ct_get_function_table();


ct_table_put(ct_func_table, CT_STATUS_HANDLER,
handle_status);
}
/**
*
* 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.
* Also, set Status to CT_AUTH_URL_ACCESS_ALLOWED forcing the
serving
* of the custom error page
*
*/
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_USER)!= NULL)
{
ct_table_put (ct_req_table, CT_URI, REGISTER_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, RE_REGISTER_USER_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;
```

```
}
```

ClearTrust SecureControl Plug-in API Reference

Hash Table Functions

The ClearTrust SecureControl Plug-in uses hash tables for both the function table and request data. The hash table functions are located in the `ct_table.h` header file. Table 3-1 lists the hash table functions:

TABLE 4-2: 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.

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.

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 4-3: Values of Input Parameter Request:

Request	Pointer To
Netscape Enterprise	Netscape Request structure, **`Request`***
IIS	Filter Context structure, **`PHTTP_FILTER_CONTEXT`**

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 in Table 3-3.

TABLE 4-4: ClearTrust SecureControl Plug-in Request Data

Key	Type	Value
`CT_ALLOWABLE_AUTH_MODES`	char *	The allowable authentication modes for the current request. The Authentication handler determines what types of authentication modes are accepted for the current URI request: `BASIC`, `CERTIFICATE`, `CUSTOM`, `EDIRECT`, `LDAP`, `NT`, `SECURID`.
`CT_AUTH_MODE`	char *	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, EDIRECT, LDAP, SECURID`–Perform Authorization only. Locate User by User ID.

TABLE 4-4: ClearTrust SecureControl Plug-in Request Data (Continued)

Key	Type	Value
CT_AUTHENTICATED	unsigned int	A bit mask that specifies the types of authentication the User is currently authenticated against: BASIC (0x00000001) CERTIFICATE (0x00000004) CUSTOM (0x00010000) EDIRECT (0x00000020) LDAP(0x00000010) NT (0x00000002) SECURID (0x00000008)
CT_DN	char *	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 *	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.
CT_FORM_AUTH_MODE	char *	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 *	Specifies whether or not the URL is protected (Yes/No).
(continued)		
CT_PASSWORD	char *	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_PREV_USER	char *	The User ID or DN for the previous authentication.
CT_STATUS	int	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 *	The requested URI. IF this value is overridden, the Web Server will be instructed to serve the new URI.

TABLE 4-4: ClearTrust SecureControl Plug-in Request Data (Continued)

Key	Type	Value
CT_USER	char *	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 *	A pointer to a buffer containing user-defined raw data included with the ClearTrust SecureControl cookie. The length of the buffer is specified in the CT_USER_DATA_LEN status code.
CT_USER_DATA_LEN	unsigned short	The length of the CT_USER_DATA buffer, in bytes. The maximum length is 1024 bytes.

NOTE: You can configure the Web Server to return a customized HTML page with the HTTP return codes.

Status Handler

The ClearTrust SecureControl Plug-in 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 3-4 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 3-3 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 4-5: Recognized Status Codes and Resulting Actions

Status Code	Resulting Actions
CT_SESSION_ACTIVE	Execute Path Check phase.
CT_AUTH_URL_PROTECTED	Execute Authentication 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 CT_SESSION_EXPIRED	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_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.

Chapter 4
Securing Non-Web Applications

This chapter tells you how to integrate non-Web Applications into the ClearTrust SecureControl security architecture. It includes the following sections:

- Integrating Non-Web Applications
- Using Non-Web APIs

Integrating Non-Web Applications

A non-Web Application is a program written in C, C++ or Java that accesses security services such as authorization, authentication and access control through ClearTrust SecureControl. These non-Web Applications can use the ClearTrust SecureControl API to fully integrate into the existing ClearTrust SecureControl security architecture. This allows non-Web programs to be managed by the same ClearTrust SecureControl security management tools that govern the rest of the system. The security policy management and control is externalized from the non-Web Application. Changes in security policy does not require re-coding or re-compiling the non-Web Applications.

For example, if an Administrator wishes to restrict a User's access to a certain set of resources, the Administrator uses the ClearTrust SecureControl manager to set the security policy. Non-Web Applications written using the ClearTrust SecureControl API then automatically pick up the changes the next time the User tries to access the secured resources. Without using the ClearTrust SecureControl API, programmers would be forced to go in and change the hard-coded setting for that User. Or, if the settings were stored in an access control list, the Administrator would have to go there to make the changes in addition to making the changes in the

ClearTrust SecureControl system. The use of the ClearTrust SecureControl API allows all security policy to be done centrally via ClearTrust SecureControl Manager.

Internal Resource Applications

Internal Resource Applications are non-Web Applications that define their functionality and resources internal to the program. A typical use of the API in this case is to allow ClearTrust SecureControl to manage the security policies of an internal resource Application. By default, when an Application is created, it has one Application Function: ACCESS. The ACCESS function is used by ClearTrust SecureControl-enabled Web Servers to determine a User's access rights to an Application; however, an Administrator may add additional Application Functions to the Application.

The additional Application Functions can further define whether or not a User has privileges to use various services associated with an Application. The access rights to these additional functions can be determined through the ClearTrust SecureControl API. By defining an Application and its functions using ClearTrust SecureControl Manager, the non-Web Application can then make calls to the ClearTrust SecureControl system to see if Users are valid and if they have been assigned access to the Applications functions.

The security policies are separate from the Application code so that the security policies can be changed without affecting the Application code. Several disparate security systems are not needed. All security polices are centrally managed under one coherent security architecture and controlled by ClearTrust SecureControl

Example. Megasoft's customer account Application has varying functionality depending on the client's service contract. Consequently, Megasoft wants all their customers to be able to access the Application but return an interface supporting only the level of functionality that matches the client's service contract. To do this, a ClearTrust SecureControl Administrator would create an Application: CustomerAccountApplication. Next, the Administrator defines the URI of the Application, `/customeraccountapp.cgi`, and a Web Server object who owns the Application, WebServer. The Administrator then defines Application

Functions representing the various functionality of the CustomerAccountApplication. For example, there could be `Create New Account`, `Edit Existing Account` and `Delete Account` functions.

The Administrator associates either a Smart Rule or Basic Entitlement with the ACCESS Application Function and each additionally defined Application Function.

During a request for the CustomerAccountApplication, the ClearTrust SecureControl enabled Web Server processes the ACCESS function to determine accessibility to the Application. Once a User is granted access, Megasoft's Application uses the ClearTrust SecureControl API to determine the different functions the customer has access rights to, and returns the correct interface which supports the function set.

Using Non-Web APIs

ClearTrust SecureControl provides an API in C and Java that allows you to use ClearTrust SecureControl as your Applications' security engine. For details about using the ClearTrust SecureControl API, refer to the **ClearTrust SecureControl Developer's Guide**.

Java API

The APIServerProxy provides the communication interface between the client Application and the ClearTrust SecureControl APIServer. Use of the Java API requires the following start-up sequence:

1. Create an APIServerProxy passing in the host computer name and port assigned to the APIServer.
2. Define the connect parameters including the name, password and role of a valid ClearTrust SecureControl Administrator. This ensures that only authorized connections are made via the API. Only programmers provided with this Administrator security information can make a valid connection.
3. After these two initialization steps, the programmer is free to call the checkAccess, checkFunction or validateUser methods. These three methods form the heart of the run time API and perform all the security access checking and User validation. The following code snippet shows how to test for access permissions to a URL.

```
APIServerProxy serverProxy = new APIServerProxy(host, port);
serverProxy.connect(userName, password, role);
```

```
if (!validateUser(userName, password)) return false;
isAccessible = checkAccess(userName, null, WebServerName,
uri);
if (isAccessible) displayMessage("Access allowed.");
else displayMessage("Access denied.”);
serverProxy.disconnect();
```

The following code snippet shows how to test for User validity and access to an Application Function.

```
APIServerProxy serverProxy = new APIServerProxy(host, port);
serverProxy.connect(userName, password, role);
if (!validateUser(userName, password)) return false;
isAccessible = serverProxy.checkFunction(userName, null,
ApplicationName, ApplicationFunction);
if (isAccessible) displayMessage("Access allowed.");
else displayMessage("Access denied.");
serverProxy.disconnect();
```

Typically, you validate the User just once before calling checkAccess or checkFunction. This saves time. If you want to validate the User each time checkAccess is called, include these parameters for the test. However, you typically connect to the server proxy only once during Application initialization.

NOTE: Error handling code is not included here.

C API

Use of the C API requires that the connect function must be called before calls to `ct_check_access`, `ct_check_function` or `ct_validate_user`. The following code snippet shows how to validate a User and check his accessibility to a secured resource.

NOTE: Error-handling code not shown.

```
ct_connect(serverName, userName, password, role, port);
ct_validate_user(userName, password, isValid);
if (!isValid) return;
ct_check_access(userName, password, WebServerName, uri,
isAccessible);
```

```
if (isAccessible) displayMessage("Access allowed.");
else displayMessage("Access denied.");
ct_disconnect();
```

How to test for user validity and access an Application Function is shown in the following code snippet:

```
ct_connect(serverName, userName, password, role, port);
ct_validate_user(userName, password, isValid);
if (!isValid) return;
ct_check_function(userName, password, ApplicationName,
functionName, isAccessible);
if (isAccessible) displayMessage("Access allowed.");
else displayMessage("Access denied.");
ct_disconnect();
```

Index