eTrust® Access Control

SDK Guide

r8 SP1

CA Product References

This document references the following CA products:

- eTrust® Access Control (eTrust AC)
- eTrust® Single Sign-On (eTrust SSO)
- eTrust® Web Access Control (eTrust Web AC)
- eTrust® CA-Top Secret®
- eTrust® CA-ACF2®
- eTrust® Audit
- Unicenter® TNG
- Unicenter® Network and Systems Management (Unicenter NSM)
- Unicenter® Software Delivery

Contact Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support.

Contents

Chapter 1: Introduction

This section contains the following topics:

About this Guide

This guide introduces you to the Application Program Interfaces (APIs) provided with eTrust AC.

eTrust AC offers several APIs for programmers who want to develop in-house eTrust AC-secured applications and to customize eTrust AC functions for specific user needs:

- For developers, the APIs supply a simple, portable interface to eTrust AC.
- For system administrators, the APIs supply a single security interface for both applications and operating systems.
- For end users, the APIs supply additional protection for their data.

eTrust AC provides sample programs. Additional examples are in the subdirectory of the directory in which eTrust AC is installed.

- For UNIX, the default directory is /opt/CA/eTrustAccessControl/apisamples.
- For Windows, the default directory is \Program Files\CA\eTrustAccessControl\SDK\samples.

APIs for UNIX and Windows

For UNIX and Windows, this version of eTrust AC includes the following APIs:

Authorization and Authentication API

The Authorization and Authentication API lets client applications request authorization for predefined or site-defined abstract resource classes using the authorization and auditing mechanisms provided by eTrust AC. Use this API to call the eTrust AC authorization daemon from within your application to check whether a user has authorization to perform the requested action.

For a detailed description, see the chapter “Authorization and Authentication API.”

Administration API

The Administration API extracts information from the eTrust AC database. This API also permits applications to perform administrative tasks such as shutting down seosd or to modify the ability to perform activities such as concurrent logins. For a detailed description, see the chapter “Administration API.”

ExitsAPI

The Exits API lets you customize the eTrust AC authorization mechanisms by complementing eTrust AC authorization routines with your own authorization routines. You can also add a special notification function to eTrust AC activities. For example, you can use this API to add a site-specific encryption algorithm to eTrust AC.

Note: (Windows only). Because eTrust AC is a certified product, you must format all object names and object property names in the UTF8 format.

APIs for UNIX Only

For UNIX alone, this version of eTrust AC includes the following API:

LogRouteAPI

The LogRoute API lets you add your own alerts to the standard eTrust AC audit log functions. You can also use the log routing daemon to add a guaranteed delivery of audit data or third-party alert systems to other programs.

Chapter 2: Authorization and Authentication API

This section contains the following topics:

Programming Guide

eTrust AC governs the user's access to a resource. Each resource belongs to a class that identifies the type of the resource. For example, records or objects of the TERMINAL class govern the user's ability to log in from a terminal. A user can access a specific resource only if the user has the permissions required to access the resource in the requested manner. For example, a user can log in from a terminal only if a rule (record) assigning the user read access to the terminal exists in the eTrust AC database. Note that the rule does not have to be an explicit assignment-you can also assign the authority using group membership or default access settings.

To learn more about eTrust AC classes and objects, and about adding user-defined resources, see the *Reference Guide* (for Windows or UNIX) and the UNIX *Utilities Guide*.

The eTrust AC function calls within the Authorization and Authentication API communicate with seosd on the local station. eTrust AC supports the following types of processes:

- Multi-user Servicing Address Space (MUSAS) applications that request authorization on behalf of other users. You normally use these applications for servers that provide services to other processes.

  Note: The terms *MUSAS* and *server* are synonymous. We use *server* throughout this guide.

- Stand-alone applications (that is, non-server applications) that request authorizations for the user currently using the application.

To use eTrust AC to protect the resources of an application, do this:

1. Add a resource class to the database. Use this resource class to protect the objects of your application.

   Note: For more information about adding a new resource class to eTrust AC, see the *seclassadm* utility in the UNIX *Utilities Guide.*

2. Add records to the application's class in the database. These records define rules for protecting your application's objects.
3. Place eTrust AC Authorization and Authentication API calls in your program.
4. Link the program with the eTrust AC library.

Both server and ordinary applications use the same library.

To use any of the eTrust AC functions, you must include the following line in your C code:

```
#include <api_auth.h>
```

The names of all the functions in the Authorization and Authentication API take the form SEOSROUTE_functionName.

This section includes sample code that demonstrates how to use some of the Authorization and Authentication API functions. Additional examples are provided in the following directories on your system drive:

- For UNIX: *eTrustACDir*/apisamples (where *eTrustACDir* is the directory you installed eTrust AC in, by default /opt/CA/eTrustAccessControl)
- For Windows: *eTrustACDir*\SDK\Samples (where *eTrustACDir* is the directory you installed eTrust AC in, by default Program Files\CA\eTrustAccessControl)

Checking the Access Authority for a User Process

Any application can use the eTrust AC Authorization and Authentication API to check whether the user can access a resource. You decide whether to perform resource access checks in your application. To write an application that uses the eTrust AC authorization mechanism, all you have to do is call a single API function called SEOSROUTE_RequestAuth with the appropriate parameters and check the return values.

The following program demonstrates how to check whether a user can access a resource:

UNIX Example

To test if eTrust AC allows you to surrogate to root by using the su root command, type the following command:

```
>upexamp SURROGATE USER.root
#include <stdio.h>
#include <string.h>
#include <memory.h>

#include “api_auth.h”

int ShowUsage(void)
{
     fprintf( stderr, “Usage:\n”
                    “ upexamp Class-Name Resource-Name\n”);
  return 1;
}

int main(int argc, char *argv[])
{ int               rv;
     char           buff[SEOSAPI_AUTH_MSGLEN];
     SEOS_ACCESS access;
     API_AUTH_RES result;     /* The result of request structure */
     if (argc != 3)
        return ShowUsage();
     memset(&access, 0, sizeof(access) );
     access.accs = SEOS_ACCS_READ;
     rv = SEOSROUTE_RequestAuth( argv[1],     /* Class Name        */
                                 argv[2],     /* Resource Name     */
                               SEOSAPI_AUTH_CURRACEE,      /*Myself*/
                               &access,
                               SEOSAPI_AUTH_LOGNONE,
                               &result,
                               buff);

     printf( “Result %s (0x%X)\n”, buff, rv );
     return 0;
}
```

After compiling and linking this example, you can check whether you have authorization to access a specific resource.

Windows Example

To test if eTrust AC lets you access a file that you were explicitly denied access to, type the following commands:

```
newfile D:\Winnt\system32\notepad.exe defaccess(all)
authorize file D:\Winnt\system32\notepad.exe \
          uid(your_UID) access(none)

>upexamp FILE D:\Winnt\system32\notepad.exe
```

The following sections describe the files used by the program.

In UNIX

The program uses the following files:

- *eTrustACDir*/apisamples/api_auth/upexamp.c
- *eTrustACDir*/apisamples/api_auth/Makefile
- *eTrustACDir*/include/api_auth.h
- *eTrustACDir*/lib/seadmapi.a

  where *eTrustACDir* is the directory you installed eTrust AC in, by default /opt/CA/eTrustAccessControl

In Windows

If you installed eTrust AC on your system, the program uses the following files:

- *eTrustACDir*\SDK\Samples\api_auth\upexamp upexamp.c
- *eTrustACDir*\SDK\Samples\api_auth\upexamp upexamp.mak
- *eTrustACDir*\SDK\Include\api_auth.h
- *eTrustACDir*\SDK\Lib\seadmapi.lib

  where *eTrustACDir* is the directory you installed eTrust AC in, by default Program Files\CA\eTrustAccessControl

Application Servers

The Authorization and Authentication API includes an interface for application servers. The server application is assumed to provide service to many users. Only server applications can perform authorization checks on behalf of users, including the user associated with the process.

The server application must perform a “pseudo-login” for each new connected client. Perform the pseudo-login by calling the SEOSROUTE_VerifyCreate function. The SEOSROUTE_VerifyCreate function provides the application with an Accessor's Entry Element (ACEE) handle for the client.

From now on, the ACEE handle returned by the SEOSROUTE_VerifyCreate function makes each call to the eTrust AC authorization check module for the client. The application should carefully maintain these handles.

The application must perform a “pseudo-logout” to release ACEE handles when a client disconnects from it or when it finishes providing service to the client. To perform the pseudo-logout, call the SEOSROUTE_VerifyDelete function. If handles are not released, both system resources and eTrust AC internal resources remain allocated to the ACEE handle. If these resources remain allocated, the unnecessary allocations can cause the system to slow down and may result in the inability to log into the system.

Only processes running under effective UID 0 (root) or users with the SERVER attribute may issue SEOSROUTE_VerifyCreate, SEOSROUTE_VerifyDelete, and SEOSROUTE_RequestAuth calls with a handle other than SEOSAPI_AUTH_CURRACEE.

Example

The following program demonstrates how to use eTrust AC to manage the security aspects of a multi-user process:

```
#include <stdio.h>
#include <string.h>
#include <memory.h>

#include “api_auth.h”

int ShowUsage(void)
{
     fprintf( stderr, “Usage:\n”
                        “ musexamp Class-Name Resource-Name
                           User-Name\n”);
     return 1;
}

int main(int argc, char *argv[])
{ int          rv;
```

```
int        usr_acee;
char       msg_buff[SEOSAPI_AUTH_MSGLEN];
SEOS_ACCESS   access;
API_AUTH_RES                 result;
/* The result of request structure */

if (argc != 4)
   return ShowUsage();

memset(&access, 0, sizeof(access) );
access.accs = SEOS_ACCS_READ | SEOS_ACCS_WRITE | SEOS_ACCS_EXEC;

rv = SEOSROUTE_VerifyCreate( argv[3], NULL, NULL, 0, NULL,
                           SEOSAPI_AUTH_LOG, &usr_acee,
                           &result, msg_buff );

if (rv)
     { printf( “Return Value: 0x%08x\n”
          “Msg: '%s'\n”, rv, msg_buff );
           return 1;
           }
else
     printf( “Got ACEE handle for user '%s': %d\n”, argv[3],
               usr_acee );

rv = SEOSROUTE_RequestAuth(argv[1],    /* Class Name         */
                    argv[2],           /* Resource Name      */
                    usr_acee,          /* User's ACEE Handle */
                    &access,
                    0,
                    &result,
                    msg_buff);

if (rv)
     printf( “Return Value: 0x%08x\n”
          “Msg: '%s'\n”, rv, msg_buff );
else
     printf( “Pass !!!!\n” );

rv = SEOSROUTE_VerifyDelete( &usr_acee, 1, msg_buff );

if (rv)
     { printf( “Return Value: 0x%08x\n”
          “Msg: '%s'\n”, rv, msg_buff );
     return 1;
  }
else
printf(“Released ACEE handle for user '%s': %d\n”,
          argv[3], usr_acee );
```

```
    return 0;
}
```

In UNIX

The program uses the following files:

- *eTrustACDir*/apisamples/api_auth/musexamp.c
- *eTrustACDir*/apisamples/api_auth/Makefile
- *eTrustACDir*/include/api_auth.h
- *eTrustACDir*/lib/seadmapi.a

  where *eTrustACDir* is the directory you installed eTrust AC in, by default /opt/CA/eTrustAccessControl

In Windows

Assuming that you installed eTrust AC on your system, the program uses the following files:

- *eTrustACDir*\SDK\Samples\api_auth\ musexamp musexamp.c
- *eTrustACDir*\SDK\Samples\api_auth\ musexamp musexamp.mak
- *eTrustACDir*\SDK\Include\api_auth.h
- *eTrustACDir*\SDK\Lib\seadmapi.lib

  where *eTrustACDir* is the directory you installed eTrust AC in, by default Program Files\CA\eTrustAccessControl

Access Authorization

The eTrust AC Authorization and Authentication API contains a single function for managing access authorizations. Use the SEOSROUTE_RequestAuth function to check whether a user has authorization to access a resource in the requested manner. For a description of how eTrust AC decides whether to grant a user access to a specific resource, see the *Administrator Guide* (for Windows or UNIX).

User Authentication

The SEOSROUTE_VerifyCreate function in the eTrust AC Authorization and Authentication API can authenticate a user. To do so, pass the existing password. In UNIX, you can also pass the new password to the API function. eTrust AC verifies that the password matches the password of the user stored in the eTrust AC database. For users defined to eTrust AC and the native operating system, eTrust AC can also use standard user accounts in the native environment to verify user passwords.

To use the SEOSROUTE_VerifyCreate function to authenticate a user, you must configure eTrust AC to enable password control and maintenance. To do this, set the PASSWORD property in the SEOS class. For example, using the selang command language, you would enter the following command:

```
setoptions class+ (PASSWORD)
```

For more information, see the setoptions command in the *Reference Guide.*

Note: When password control is enabled, the system administrator can take advantage of its format restrictions, aging, and history maintenance.

Because eTrust AC does not use the native operating system password, you can create user accounts that can use only servers protected by eTrust AC. These accounts are not valid UNIX or Windows accounts. In UNIX, these users do not have direct access to stations that enable UNIX shell sessions. In Windows, they cannot log in through a terminal. Therefore, these users are not able to log in interactively in either environment.

Managing Error Messages

The Authorization and Authentication API includes a function that helps you manage error messages. When an eTrust AC function fails, it returns an error code. The function also writes the error message describing the reason for the failure in the szMsg parameter. The error message (szMsg) is listed for each API.

Note: The SEOSROUTE_ParseApiError function analyzes the error message stored in the szMsg parameter and returns the integer representing the error code. This function is included to provide more information (the real error code from seosd) in case SEOSAPI_AUTH_REMOTE_ERR is returned. This would indicate some error in seosd.

Compiling and Linking with the Library

The procedures for compiling and linking in UNIX and in Windows are described in the following sections.

Compiling an Application in UNIX

To compile your application with the eTrust AC library, include the api_auth.h header file in the C source code that calls the library. You can use any ANSI-C compliant compiler.

Linking Your Application in UNIX with the AuthAPI Library

The method used to link your application to the AuthAPI library depends on the operating system you are using, as shown in the following table. Use these lines together with the rest of the command used by your operating system for linking an application.

The examples in the following table assume that eTrust AC was installed in the /opt/CA/eTrustAccessControl directory. The libraries are therefore assumed to be located in the/opt/CA/eTrustAccessControl/lib directory and the header files in the /opt/CA/eTrustAccessControl/include/API directory.

Platform	**Command**
IBM AIX	`cc sample.c -I/opt/CA/eTrustAccessControl/include \` `-bI:/opt/CA/eTrustAccessControl/lib/SEOS_binder.exp \` `/opt/CA/eTrustAccessControl/lib/seadmapi.a -o sample`
All other platforms	`cc sample.c -I/opt/CA/eTrustAccessControl/include \` `/opt/CA/eTrustAccessControl/lib/seadmapi.a -o sample`

Note: The examples shown in this table are shipped with a makefile that you can use to set the various flags required by each environment.

Compiling and Linking an Application in Windows

Each sample has a Microsoft makefile. To compile the programs, enter one of the following commands:

```
NMAKE /f “makefile name“ CFG=“source C code - Win32 Release”
```

or

```
NMAKE /f “makefile name“
```

For example, to build the sample_FetchList sample, change to the sample_FetchList subdirectory:

```
(..Program Files\CA\eTrustAccessControl\SDK\Samples\seadmapi\
   sample_FetchList)
```

Then type:

```
NMAKE /f “sample_FetchList.mak”
```

Note: Before running the nmake command, run the VCVARS32.BAT file from the Visual C++ BIN directory to set the compiler environment variables, and ensure that the nmake.exe is in your system variable path.

Remote Authorization API

The Remote Authorization API lets a client application perform an authorization check against a remote authorization server. This is useful for a site that has an eTrust AC authorization server and clients-not necessarily running eTrust AC-running applications that require an authorization check from the server. This is done in three phases.

1. Login phase. The remote authorization API invokes the lca_rmtauth_Init function that performs a login to the remote server. It does this by supplying the password of the user for whom the client application runs, and checking it against the user's password on the remote server.
2. Authorization request phase. In this phase, the API invokes the lca_rmtauth_CheckAccess function that sends the server all the information needed to check authorization. This includes the user name, the class, the object, and the access type. The function returns a structure of type API_AUTH_RES that includes the request result. This function is a remote version of the SEOSROUTE_RequestAuth function.
3. Termination phase. Here the API calls the lca_Terminate function that closes the connection to the server and frees any allocated memory.

Files

The remote authorization API uses the following files:

- *eTrustACDir*/apisamples/api_auth/rmt_auth.c
- *eTrustACDir*/apisamples/api_auth/Makefile
- *eTrustACDir*/include/api_auth.h
- *eTrustACDir*/include/langapi.h

  where *eTrustACDir* is the directory you installed eTrust AC in, by default /opt/CA/eTrustAccessControl

For details on the remote authorization API functions, see the chapter "Language Client API."

Example

The following program demonstrates how a client application can perform a remote authorization check:

```
#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <unistd.h>

#include "api_auth.h"
#include "langapi.h"

int ShowUsage(void)
{
  fprintf( stderr, "Usage:\n"
     " rmt_auth User-Name Class-Name Resource-Name"
     "Access"
  return 1;
}

int main(int argc, char *argv[])
{
char *output;
int rv;
API_AUTH_RES result;
char input[200];
char *Passwd = NULL;

    if ( argc != 6 )
        return ShowUsage();
    Passwd = getpass(“Please enter your password:”);
    if ( Passwd == NULL )
    {
       printf(“Illegal password!!\n”);
       return 1;
}

    /*
     * Initialization. get the host name and password from the
     * command line parameters
     */
    rv = lca_rmtauth_Init(“rmt_auth”, &output, argv[5], Passwd);
    if ( rv )
    {
        printf( “Return value: 0x%08x\n”
              “%s\n”, rv, (output) ? output : “Initialization failed” );
        return 1;
    }
    Passwd = NULL;

    /*
     * Send the request to the server and get the result
     */
rv = lca_rmtauth_CheckAccess(argv[1], argv[2], argv[3], argv[4], &result);
    if ( rv )
    {
        printf(“Return value: 0x%08x\n”
               “Remote Access Check failed\n”, rv);
        return 1;
    }

    /*
     * Print the results.
     * Further analysis as for the reasons for granting/denying
     * access can be made at this stage
     */
    printf(“%s's access %s to %s %s at %s is %s!!\n”,
               argv[1], argv[4], argv[2], argv[3], argv[5],
            ((char) result.result == 'P') ? “Permitted” : “Denied”);

    lca_Terminate();
}
```

After compiling and linking this example, you can check whether a user is authorized to access a specific resource. For example, to see whether user Joe has read access to file /etc/passwd, type the following command:

`rmt_auth Joe FILE /etc/passwd Read` *`authorization-server-name`*

You are prompted for your password, after which you receive the reply.

Authorization and Authentication API Functions

The Authorization and Authentication API consists of the following functions:

SEOSROUTE_ParseApiError

Converts an error string into the integer representing the error code.

SEOSROUTE_RequestAuth

Checks whether a user is authorized to access a resource using the specified access type.

SEOSROUTE_VerifyCreate

Creates an ACEE handle for a user.

SEOSROUTE_VerifyDelete

Deletes a user's ACEE handle.

SEOSROUTE_ParseApiError Function

The SEOSROUTE_ParseApiError function parses the error string returned by the verification and authentication functions and returns the integer value associated with it.

Notes:

- Any user can call the SEOSROUTE_ParseApiError function.
- This function is included so you can get the real error code from seosd if the value returned by SEOSROUTE_RequestAuth, SEOSROUTE_VerifyCreate, or SEOSROUTE_VerifyDelete is SEOSAPI_AUTH_REMOTE_ERR.

```
int SEOSROUTE_ParseApiError(const char *szErrMsg);
```

szErrMsg

The error string returned by eTrust AC in the *szMsg parameter

SEOSROUTE_RequestAuth Function

The SEOSROUTE_RequestAuth function asks seosd whether the specified user is allowed to access the specified resource using the specified access type.

The SEOSROUTE_RequestAuth function sends the request to seosd, which first checks whether the parameters are valid. If they are, seosd performs its standard resource authorization check: Is the specified user authorized to access the specified resource in the specified way?

The first five parameters must be supplied by the application; the last two parameters are returned by eTrust AC and can be used by the application to make decisions and provide the user with status information.

The function returns an integer that takes on one of the following values:

SEOSAPI_AUTH_OK

The user is allowed to access the resource as requested.

SEOSAPI_AUTH_BADACCESS_ERR

An invalid access authority was specified.

SEOSAPI_AUTH_DENY

The request was denied.

SEOSAPI_AUTH_NORESPONSE_ERR

The seosd daemon is not responding.

SEOSAPI_AUTH_NOTROOT_ERR

The user ID of the calling process is not 0 (root), and the user executing the calling process does not have the SERVER attribute.

SEOSAPI_AUTH_REMOTE_ERR

The daemon or service returned an error described in szMsg.

Notes:

A server application normally calls SEOSROUTE_VerifyCreate before calling SEOSROUTE_RequestAuth, to get an ACEE handle for the user whose authorization is being checked.

- When SEOSROUTE_RequestAuth is called with the hACEE parameter set to SEOSAPI_AUTH_CURRACEE, authorization is requested for the user executing the calling process. eTrust AC does not check whether the user has the SERVER attribute. In UNIX, eTrust AC does not check whether the process is running under the effective user ID of root.
- All users can use the SEOSROUTE_RequestAuth function. Only a server process can use the query for an ACEE other than its own.

```
int SEOSROUTE_RequestAuth(const char    *szClass,
                          const char    *szEntity,

                          int           hACEE,
                          SEOS_ACCESS   *pAccess,
                          int           LogOpt,
                          API_AUTH_RES  *pRes,
                          char          *szMsg);
```

szClass

The name of the class to which the resource belongs.

szEntity

The name of the record, or object, representing the resource being accessed.

Note: The case of the szEntity parameter is important when the szClass supports case-sensitive objects.

hACEE

The ACEE handle of the accessor. To specify the ACEE of the user associated with current process, specify SEOSAPI_AUTH_CURRACEE. Specifying an ACEE handle other than SEOSAPI_AUTH_CURRACEE requires the user associated with the calling process to have the SERVER attribute or the calling process to be running under the effective user ID of 0 (root).

pAccess

A pointer to a structure containing the requested access. The structure contains the single data member access of type SEOS_ACCS. Valid values for this member are:

- SEOS_ACCS_ALL
- SEOS_ACCS_CHMOD
- SEOS_ACCS_CHOWN
- SEOS_ACCS_DELETE
- SEOS_ACCS_EXECUTE
- SEOS_ACCS_NONE
- SEOS_ACCS_READ
- SEOS_ACCS_RENAME
- SEOS_ACCS_SEC
- SEOS_ACCS_UPDATE
- SEOS_ACCS_UTIME
- SEOS_ACCS_WRITE.

LogOpt

A flag that determines whether an audit log entry must be made. Valid values are:

SEOSAPI_AUTH_LOGNONE

For regular users, which are not root, if the authorization request succeeds, do not create an audit record. If the authorization request fails, create an audit record if the current rules in the database require auditing.

For server applications and for root users, do not create an audit record, regardless of whether the authorization request succeeds or fails (value = 0).

SEOSAPI_AUTH_LOG

If the current rules in the database require it, create an audit record (value = 1).

SEOSAPI_AUTH_LOGALL

For regular users, this is an invalid option and is mapped to SEOSAPI_AUTH_LOG.

For server applications, always create an audit record regardless of the database rules (value = 2).

SEOSAPI_AUTH_LOGFAIL

For regular users, this is an invalid option and is mapped to SEOSAPI_AUTH_LOG.

For server applications, create an audit record only if the authorization request fails and the database rules require it (value = 3).

SEOSAPI_AUTH_LOGNONE_USER

For regular users (including root), if the authorization request succeeds, do not create an audit record. If the authorization request fails, create an audit record if the current rules in the database require auditing.

For server applications, do not create an audit record, regardless of whether the authorization request succeeds or fails (value = 4).

SEOSAPI_AUTH_LOGNEVER

Do not create an audit record (value = 5).

pRes

A pointer to the API_AUTH_RES structure containing the authorization result. For more information about the API_AUTH_RES structure, see API_AUTH_RES in this chapter.

szMsg

A pointer to a buffer SEOSAPI_AUTH_MSGLEN into which eTrust AC returns a status message.

More information:

SEOSROUTE_VerifyCreate Function (see page 37)

SEOSROUTE_RequestAuthAzn Function

The SEOSROUTE_RequestAuthAzn function sends seosd an authorization request. eTrust AC first checks whether the parameters are valid. If they are, eTrust AC performs its standard resource authorization check realized in function SEOSROUTE_RequestAuth. It also processes eTrust Web AC with user attributes.

If the function succeeds, the return value is zero. Otherwise, the return value is an error code.

```
int SEOSROUTE_RequestAzn (
     LPCSTR szClass,
     LPCSTR szEntity,
     LPCSTR szAccess,
     HANDLE *phUserAttributes,
     int hACEE,
     int LogOpt,
     API_AZN_RES  *pres,
     API_RESP_TAB **response,
     LPCSTR szMsg
);
```

hACEE

The ACEE handle of the accessor. For more information, see SEOSROUTE_RequestAUTH in this chapter.

LogOpt

A flag that determines whether an audit log entry must be made. For more information, see SEOSROUTE_RequestAUTH in this chapter.

phUserAttributes

A pointer to a hash table created by the function SEOSROUTE_Create RequestAzn. This pointer is related only to the SEOSD process.

pRes

A pointer to the API_AZN_RES structure containing the authorization result. For more information about the API_AZN_RES structure, see API_AZN_RES in this chapter.

response

A pointer to the buffer containing the response list.

SzAccess

A pointer to a null terminated string that specifies the access name corresponding to the object described by szEntity.

SzClass

A pointer to a null terminated string containing the name of the resource class.

SzEntity

A pointer to a null terminated string containing the name of the object representing the resource being accessed.

Note: The case of the szEntity parameter is important when the szClass supports case-sensitive objects.

szMsg

A pointer to a buffer SEOSAPI_AUTH_MSGLEN bytes long into which eTrust AC returns a status message.

More information:

API_AZN_USERATTR Structure (see page 36)
SEOSROUTE_CloseRequestAzn Function (see page 36)

SEOSROUTE_CreateRequestAzn Function

The SEOSROUTE_CreateRequestAzn function sends seosd an array of the user attributes relating to same user and receives pointers specifying a hash table created on the user attributes base.

This handle is used in other eTrust WebAC API functions.

Note: It is important to call the SEOSROUTE_CloseRequestAzn function after processing a user.

If the function succeeds, the return value is zero. Otherwise, the return value is an error code.

```
INT SEOSROUTE_CreateRequestAzn(

     LPCSTR szUserDir,
     DWORD cEntries,
     LPAPI_AZN_USERATTR pUserAttr,
     PHANDLE phUserAttributes,
     LPSTR szMsg
);
```

SzUserDir in

Pointer to a null terminated string specifying the user directory.

CEntries

Value specifying the number of the API_AZN_USERATTR structures in array pointed to by pUserAttr. Cannot be zero (0).

PUserAttr in

Pointer to an array containing user attributes.

PhUserAttributes out

If the function succeeds, the parameter specifies a pointer to a hash table calculated using user attributes.

If the function fails, the value is INVALID_HANDLE_VALUE.

SzMsg out

A pointer to a buffer SEOSAPI_AUTH_MSGLEN bytes long into which eTrust AC returns a status message.

More information:

API_AZN_USERATTR Structure (see page 36)
SEOSROUTE_CloseRequestAzn Function (see page 36)
SEOSROUTE_RequestAuthAzn Function (see page 33)

SEOSROUTE_CloseRequestAzn Function

The SEOSROUTE_CloseRequestAzn function removes the hash table pointer by hUserAttributes. After calling this function, SEOSD cannot process authorization requests for the specified user unless SEOSROUTE_CreateRequestAzn( ) is processed again.

If the function succeeds, the return value is zero. Otherwise, the return value is an error code.

```
INT SEOSROUTE_CloseRequestAzn(
     HANDLE hUserAttributes,
     LPSTR szMsg
);
```

HUserAttributes in

A pointer to a hash table calculated using user attributes.

SzMsg out

A pointer to a buffer SEOSAPI_AUTH_MSGLEN bytes long into which eTrust AC returns a status message.

More information:

API_AZN_USERATTR Structure (see page 36)
SEOSROUTE_CreateRequestAzn Function (see page 35)
SEOSROUTE_RequestAuthAzn Function (see page 33)

API_AZN_USERATTR Structure

The API_AZN_USERATTR structure specifies attribute information.

```
typedef  struct tagAPI_AZN_USERATTR {
     char szAttName[ONAME_SIZE];
     char szAttVal[ONAME_SIZE];
}API_AZN_USERATTR, *PAPI_AZN_USERATTR, FAR *LPAPI_AZN_USERATTR;
```

szAttName

Null terminated string containing the attribute name.

szAttVal

Null terminated string containing the attribute value.

More information:

SEOSROUTE_CloseRequestAzn Function (see page 36)
SEOSROUTE_RequestAuthAzn Function (see page 33)

SEOSROUTE_VerifyCreate Function

The SEOSROUTE_VerifyCreate function performs a pseudo-login to eTrust AC and returns the ACEE handle that is created.

The first six parameters must be supplied by the application; the last three are returned by eTrust AC and can be used by the application to make decisions and provide the user with status information.

The function returns an unsigned integer that is one of the following values:

SEOSAPI_AUTH_OK

The user is allowed to access the resource as requested.

SEOSAPI_AUTH_BADPASSWD_ERR

The password does not match the expected password.

SEOSAPI_AUTH_DENY

The request was denied.

SEOSAPI_AUTH_NORESPONSE_ERR

Seosd is not responding.

SEOSAPI_AUTH_NOTROOT_ERR

The user ID of the calling process is not 0 (root), and the user executing the calling process does not have the SERVER attribute.

SEOSAPI_AUTH_NOUSERID_ERR

A user name was not supplied.

SEOSAPI_AUTH_NULLACEE_ERR

The phACEE parameter is a null pointer.

SEOSAPI_AUTH_REMOTE_ERR

The eTrust AC daemon returned an error described in szMsg.

Notes:

- The SEOSROUTE_VerifyCreate function should be used only by a multi-user (server) process that performs services on behalf of other users. A single-user process can call the SEOSROUTE_RequestAuth function without first calling the SEOSROUTE_VerifyCreate function.
- To execute the SEOSROUTE_VerifyCreate function, the calling process must have an effective user ID of 0 or the user associated with the calling process must have the SERVER attribute.

```
int SEOSROUTE_VerifyCreate(const char   *szUserId,

                           const char   *szPwd,
const char   *szNewPwd,
int          bPwdChk,
const char   *szTerm,
int          LogOpt,
int          *phACEE,
API_AUTH_RES *pRes,
char         *szMsg);
```

szUserId

The name of the user for whom the ACEE is created. This parameter must be supplied.

szPwd

The password of the user identified by szUserId. If a NULL pointer is specified, eTrust AC skips the password check.

szNewPwd

The new password, should your application be changing the user's password in the eTrust AC database. Specify a NULL pointer if you are not specifying a new password or if a NULL pointer is specified for szPwd.

bPwdChk

A flag that determines whether the password is to be checked or not. You can combine the following flag values (using bitwise OR):

VERCRE_CHECK_CUR

Check that the current password is valid.

VERCRE_CHECK_NEW

In UNIX, check that the new password is valid according to active password policy rules.

VERCRE_CHECK_QUICKLOGIN

Simulate login without checking for time restrictions.

szTerm

The name of the terminal from which the user logged onto the system.

LogOpt

A flag that determines whether an audit log entry must be made. Valid values are:

SEOSAPI_AUTH_LOGNONE

For server applications, do not create an audit record, regardless of whether the authorization request succeeds or fails (value = 0).

SEOSAPI_AUTH_LOG

If the current rules in the database require it, create an audit record (value = 1).

SEOSAPI_AUTH_LOGALL

For server applications, always create an audit record regardless of the database rules (value = 2).

SEOSAPI_AUTH_LOGFAIL

For server applications, create an audit record only if the authorization request fails and the database rules require it (value = 3).

phACEE

The ACEE handle returned by eTrust AC. This value is used by eTrust AC in subsequent authorization checks for the currently verified user.

pRes

A pointer to the API_AUTH_RES structure containing the authorization result. For more information, see API_AUTH_RES in this chapter.

szMsg

A pointer to a buffer SEOSAPI_AUTH_MSGLEN bytes long into which eTrust AC returns a status message.

More information:

SEOSROUTE_VerifyDelete Function (see page 40)

SEOSROUTE_VerifyDelete Function

The SEOSROUTE_VerifyDelete function releases an ACEE. Use this function to release ACEEs created using the SEOSROUTE_VerifyCreate function. Your application should release ACEEs once they are no longer required, because each allocated handle uses system resources and eTrust AC internal resources. These resources are limited.

The first two parameters must be supplied by the program; the last parameter is returned by eTrust AC and can be used by the program to make decisions and provide the user with status information.

The function returns an integer that is one of the following values:

SEOSAPI_AUTH_OK

The ACEE was released.

SEOSAPI_AUTH_NOACEE_ERR

The ACEE handle was not found.

SEOSAPI_AUTH_NORESPONSE_ERR

Seosd is not responding.

SEOSAPI_AUTH_NOTROOT_ERR

The user ID of the calling process is not 0 (root), and the user executing the calling process does not have the SERVER attribute.

SEOSAPI_AUTH_NULLACEE_ERR

The phACEE parameter is a NULL pointer.

SEOSAPI_AUTH_REMOTE_ERR

The eTrust AC daemon returned an error described in szMsg.

Note: To execute the SEOSROUTE_VerifyDelete function, the calling process must have an effective user ID of 0 or the user associated with the calling process must have the SERVER attribute.

```
int SEOSROUTE_VerifyDelete(int  *phACEE,
                           int  bLog,

                           char *szMsg)
```

phACEE

A pointer to the handle of the ACEE to be released.

bLog

A flag that determines whether an audit log entry is created. It can have a value of 0 or 1. To create a log entry, set the bLog parameter to 1.

szMsg

A pointer to a buffer SEOSAPI_AUTH_MSGLEN bytes long into which eTrust AC returns a status message.

More information:

SEOSROUTE_VerifyCreate Function (see page 37)

Structures and Data Types

This section describes the data structures used by the Authorization and Authentication API functions to pass information back and forth between the functions and the eTrust AC daemons (in UNIX) and services (in Windows). Every field of each data structure is described.

The Authorization and Authentication API functions use the following structures:

API_AUTH_RES

Holds the result of an authorization check.

SEOS_ACCESS

Encapsulates a single member of type SEOS_ACCS.

SEOS_ACCS

Holds a list of access flags.

SEOS_OID

Holds an object identification descriptor.

API_AUTH_RES Structure

The API_AUTH_RES structure holds the results of an authorization check.

int result

A code indicating the result of the authorization check. Valid values are:

- **P**-Requested access to resource was granted.
- **D**-Requested access to resource was denied.
- **C**-The access request check was incomplete.

int last_stage

The authorization stage at which the information in the structure was written. This information is useful if access was granted but the authorization failed later for some reason.

int grant_stage

The authorization stage at which the permit or deny decision was made.

SEOS_ACCS accs

An unsigned long integer representing the type of access requested. For a list of possible values, see SEOS_ACCS in this chapter.

SEOS_OID oidRes

The object ID of the resource for which authorization was checked.

SEOS_OID oidGroup

If accumulated group rights are being checked and if access is allowed or denied by a group, this member stores the object ID of the last group checked.

If accumulated group rights are not being checked and if access is allowed or denied by a group, this member stores the object ID of the group.

API_AZN_RES Structure

The API_AZN_RES structure holds the results of an authorization check.

int result

A code indicating the result of the authorization check. Valid values are:

- **P**-Permission to access the resource was granted.
- **D**-The requested access was denied.
- **C**-The database has been corrupted.

int last_stage

The authorization stage at which the information in the structure was written. This information is useful if access was granted but the authorization failed later for some reason.

int grant_stage

The authorization stage at which the permit or deny decision was made.

SEOS_ACCS accs

An unsigned long integer representing the type of access requested. For a list of possible values, see SEOS_ACCS in this chapter.

SEOS_OID oidRes

The object ID of the resource for which authorization was checked.

SEOS_OID oidGroup

If accumulated group rights are being checked and if access is allowed or denied by a group, this member stores the object ID of the last group checked.

If accumulated group rights are not being checked and if access is allowed or denied by a group, this member stores the object ID of the group.

SEOSDB_CDF dfRespTab

Structure representing the class definition of a record in the database.

SEOSDB_PDF pdfRespTab

Structure representing the property definition of a record in the database.

SEOSDB_ODF odfRespTab

Structure representing the object definition of a record in the database.

SEOS_ACCESS Structure

The SEOS_ACCESS structure encapsulates a single member of the type SEOS_ACCS.

SEOS_ACCS accs

An unsigned long integer representing the type of access requested. A list of possible values is detailed for the SEOS_ACCS structure *(see page* 45*)*.

SEOS_ACCS Structure

The SEOS_ACCS data type is an unsigned long integer representing the type of access requested.

Access types currently defined for the APIs include the following:

For All Requests

SEOS_ACCS_ANY

Everything is allowed.

SEOS_ACCS_AUTHORIZE

Changing ACLs is allowed.

SEOS_ACCS_CREATE

Creating new files in class FILE and new objects in class ADMIN is allowed.

SEOS_ACCS_DELETE

Deleting is allowed (same as SEOS_ACCS_ERASE).

SEOS_ACCS_ERASE

Deleting is allowed.

SEOS_ACCS_EXEC

Executing programs is allowed.

SEOS_ACCS_FILESCAN

Scanning files is allowed.

SEOS_ACCS_JOIN

Adding users to groups or removing users from groups is allowed.

SEOS_ACCS_MODIFY

Renaming is allowed.

SEOS_ACCS_NONE

Nothing is allowed.

SEOS_ACCS_PASSWD

Changing password attributes is allowed.

SEOS_ACCS_READ

Read access is allowed.

SEOS_ACCS_RENAME

Renaming files is allowed.

SEOS_ACCS_WRITE

Write access is allowed.

SEOS_ACCS_reserved

Not used.

For UNIX Requests Only

SEOS_ACCS_CHOWN

Changing ownership is allowed.

SEOS_ACCS_CHGRP

Changing group setting is allowed.

SEOS_ACCS_CHMOD

Changing file mode is allowed.

SEOS_ACCS_UTIMES

Changing modification time of files is allowed.

Generic Attributes

SEOS_ACCS_SEC

Changing ACLs of files is allowed.

Macros for Multiple Access Requests

SEOS_ACCS_CHOG

CHOWN + CHGRP

SEOS_ACCS_UPDATE

READ + WRITE + EXEC

SEOS_ACCS_CONTROL

CHOG + CHMOD + UTIMES + SEC + UPDATE

Note: For possible additional values for this field, see the file *eTrustACDir*/include/seostype.h.

SEOS_OID Data Type

The SEOS_OID data type is an unsigned long integer representing the object ID of a record in the database.

Each object in the database has a unique object ID. If you know the object ID, you can use seadmapi to retrieve information about the object.

Chapter 3: Exits API

This section contains the following topics:

Programming Guide

The Exits API lets you insert your own functions to be executed just before or after eTrust AC authorizes a requested activity. The seosd daemon/service monitors all system, program, and user activities. It intercepts every activity and decides whether to authorize the requested action. You can insert your own registered functions just before (pre) or after (post) eTrust AC makes these decisions.

For example, you can register a pre-exit function for execution before eTrust AC considers each login request. Your exit function gains control just before eTrust AC starts to authorize a login request. After completing its task, your exit function returns control to eTrust AC with a return code indicating your function's authorization decision. Your function must return one of the following return codes:

SEOS_EXITR_CHECK

Instructs eTrust AC to perform its own standard authorization check.

SEOS_EXITR_PASS

Instructs eTrust AC to grant the request. eTrust AC does not perform its own standard authorization check.

SEOS_EXITR_DENY

Instructs eTrust AC to deny the request. eTrust AC does not perform its own standard authorization check.

If the decision is SEOS_EXITR_PASS or SEOS_EXITR_DENY, eTrust AC grants or denies the request immediately. If it is SEOS_EXITR_CHECK, eTrust AC continues with its own standard authorization check. System, program, and user activities that require authorization by eTrust AC are called *events*. eTrust AC authorizes five categories of events:

- Login
- General resource check
- TCP/IP request (available in UNIX only)
- Password quality check
- Password change

These events are described in eTrust AC Events in this chapter. Compiling and linking procedures are described in Compiling and Linking in this chapter.

Creating a New Exit Function

In UNIX

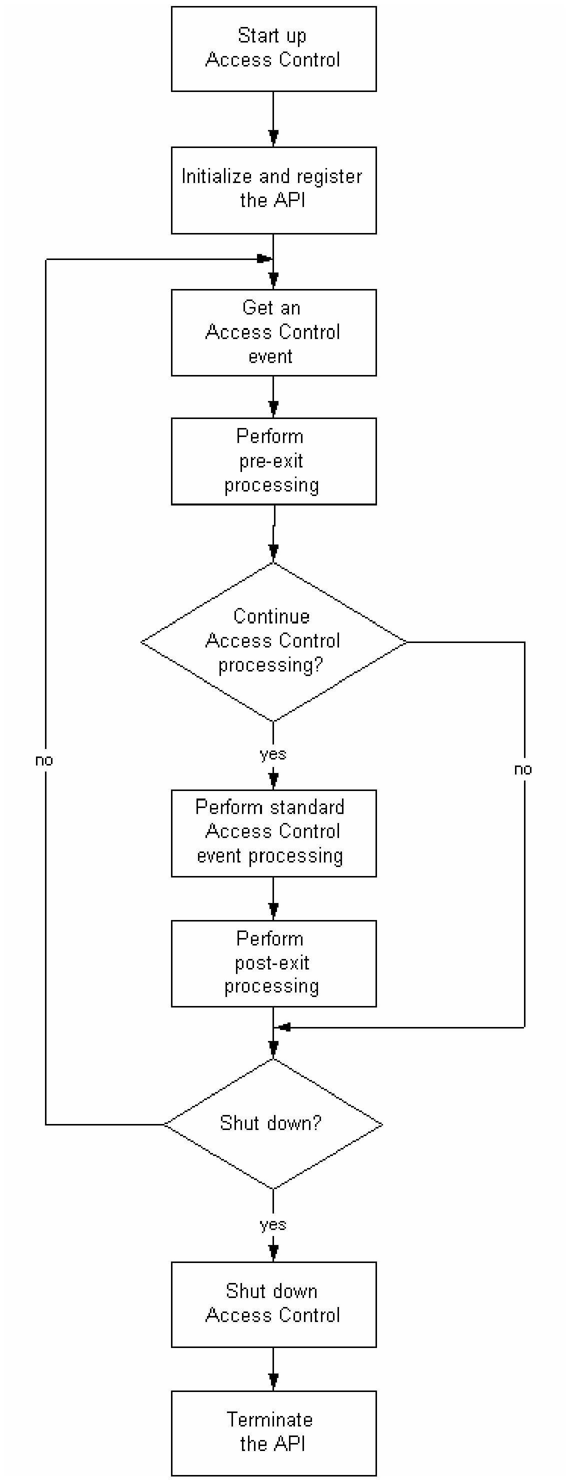
New exit functions are added to the seosd and sepass programs by writing C-language functions that can be compiled and linked to a shared library. The seos.ext and sepass.ext files must be changed to include this new, shared library.

An Exits API function has three parts:

- Registration
- Implementation
- Termination

The registration function initializes your Exits API function and registers it with the eTrust AC programs. The implementation function adds your tasks to the standard eTrust AC processing. The termination function unregisters and shuts your program down properly when the eTrust AC programs themselves terminate.

The following diagram illustrates the flow of the Exits API initialization, implementation, and termination functions in UNIX.


Start up Access Control
Initialize and register the API
Get an Access Control event
Perform pre-exit processing
Continue Access Control processing?
yes
no
Perform standard Access Control event processing
Perform post-exit processing
Shut down?
no
yes
Shut down Access Control
Terminate the API

Your Exits API exit functions take advantage of functions and header files provided by eTrust AC. You use the same registration, initialization, and termination functions for all your exit functions, whether they link to seosd or sepass. For more information about the predefined functions used in an exit function, see Exits API Functions in this chapter.

When your Exits API function is ready, you must link your new API to the eTrust AC daemons. For information about compiling and linking procedures, see Compiling and Linking.

In UNIX for Utilities

You add new exit functions to the *sesudo* and *sesu* programs by writing C-language functions that can be compiled and linked to a shared library. You must change the *.ext files to include this new, shared library.

An Exits API function has three parts:

- Registration
- Implementation
- Termination

The registration function initializes your Exits API function and registers it with the utilities. The implementation function adds your tasks to the standard utilities' regular activities. The termination function unregisters and shuts your program down properly when the utilities themselves terminate.

Note: Examples for utility exits APIs are available in the following directory:

```
eTrustACDir/apisamples
```

In Windows

You add new exit functions to the seosd service and to pwdchange.dll by writing C-language functions that can be compiled and linked to a dynamic link library (dll). To install new exits, you must add new sub-keys under the following registry key:

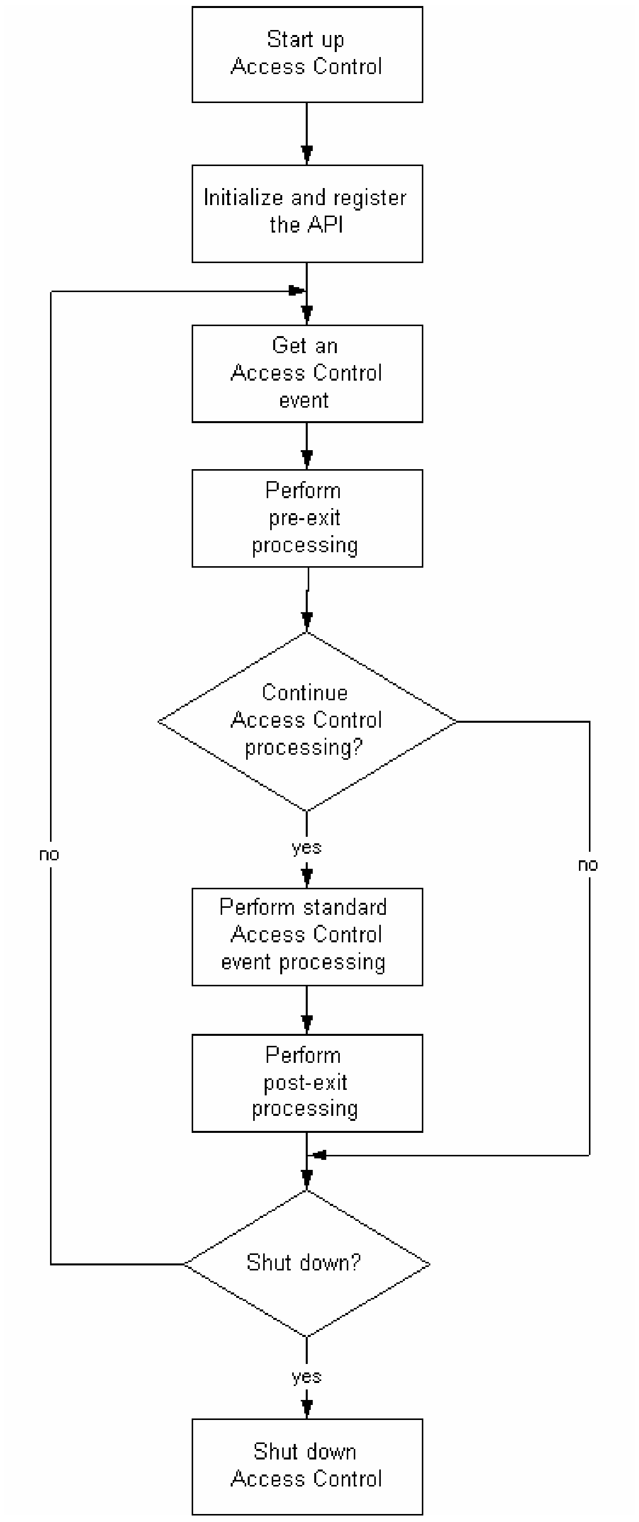
```
HKEY_LOCAL_MACHINE\Software\ComputerAssociates\eTrustAccessConrtrol\Exits\Engine
```

Exit APIs can fall into one of the following categories:

- Registration callback functions
- Implementation callback functions

The registration function initializes your Exits API function and registers it with eTrust AC programs. The implementation function adds your tasks to the standard eTrust AC processing.

The following diagram illustrates the flow of the Exits API initialization and implementation.


Start up Access Control
Initialize and register the API
Get an Access Control event
Perform pre-exit processing
Continue Access Control processing?
yes
no
Perform standard Access Control event processing
Perform post-exit processing
Shut down?
no
yes
Shut down Access Control

Your Exits API exit functions take advantage of functions and header files provided by eTrust AC. You use the same registration and implementation functions for all your exit functions, whether they link to seosd or pwdchange.

When your Exits API function is ready, you must link your new API to the eTrust AC services. For information about compiling and linking procedures, see Compiling and Linking.

Data Structures

All Exits API functions use special data structures provided by eTrust AC to pass information back and forth between functions. Programmers must know the specific formats and data types used by these structures to access them correctly in their own programs. For information about these formats and data types, see Structure and Data Types in this chapter. The input data structure used by your exit function depends on the event being intercepted by the function. All functions use the same output data structure. The following table lists the data structures used by the Exits API functions.

Event	**Data Structure**	**Type**
Login	SEOS_EXITLOGIN	Input
General resource check	SEOS_EXITGENR	Input
TCP/IP request (for UNIX only)	SEOS_EXITINET	Input
Password quality check Password change	SEOS_EXITPASS	Input
All events	SEOS_EXITRES	Output

If your Exits API function is successful, it should fill in the SEOS_EXITRES structure and return 0. When eTrust AC receives a return code of 0, eTrust AC checks the result field in the SEOS_EXITRES structure. If the SEOS_EXITRES value is Pass or Deny, it is acted on immediately, and eTrust AC does not execute its own authorization check. If the result is Check, eTrust AC continues with its own authorization check.

If your Exits API function fails, it should fill in the SEOS_EXITRES structure and return a nonzero error code. When eTrust AC receives a nonzero return code, it adds an entry to the error log file with the source file name and line number as they appear in the SEOS_EXITRES structure. The other values set in SEOS_EXITRES are ignored. eTrust AC then continues with its own authorization check.

This chapter provides two sample Exits API functions. These examples can help you get started with your own programs. The first example is a simple counter that intercepts every eTrust AC authorization call and keeps statistics on how often such calls are made. The second example adds a new restriction to the password authorization algorithm of eTrust AC. This exit function stops users from choosing the word *password* as their new password.

See System Design and Limits in this chapter for more information about:

- The modular design used by eTrust AC that your functions should maintain
- The limits imposed by eTrust AC on Exits API functions
- How to avoid compile and run-time errors

eTrust AC Events

System, program, and user activities that require authorization by eTrust AC are called events. Events are grouped into five categories:

- Login
- General resource check
- TCP/IP request (UNIX only)
- Password quality check
- Password change

Exit functions for password quality check and password change events are linked to the password utility *sepass* in UNIX and the password dll pwdchange in Windows. Exit functions for login, general resource check, and TCP/IP request events are linked to *seosd*.

Events Linked to seosd

The following events are registered with seosd:

- Login
- General resource check
- TCP/IP request (UNIX only)

A *login* event occurs whenever a user attempts to log in to the system. All information relevant to the login attempt is passed to the API function. This information includes:

- The user name and user ID of the user involved
- The terminal from which the user is trying to log in
- The device and inode numbers, and the name of the program attempting to perform the login

This information is passed to the Exits function in the SEOS_EXITLOGIN structure.

Note: Part of the login authorization process involves a check of whether the user is allowed to log in from the terminal from which the login request is received. If a general resource exit function is registered, that exit function is called as part of the login check.

A *general resource check* event occurs whenever eTrust AC checks the authorization for any system request except login and TCP/IP requests. All information relevant to the system request is passed to the API function. This information includes:

- The class and resource name of the object involved
- The user ID, user handle, and user name of the user involved
- The device and inode numbers, and the name of the program involved
- The terminal from which the user is submitting the request
- The type of access requested

This information is passed to the function in the SEOS_EXITGENR structure.

In UNIX, a *TCP/IP request* event occurs whenever a remote host attempts to connect to the local host. In this case, no information is available on the specific user. All information relevant to the connection attempt is passed to the API function. This information includes the host address and name, the type of access requested, the name of the program involved, the port number, and the protocol code. The information is passed to the function in the structure SEOS_EXITINET.

Events Linked to sepass in UNIX

The *password quality check* and *password change* events are registered with the password utility *sepass*. The Exits API data structure SEOS_EXITPASS is used to pass information about these events between functions. For more information about the SEOS_EXITPASS data structure, see Structure and Data Types in this chapter.

A *password quality check* event occurs whenever a user attempts to enter a new user password. eTrust AC always calls the verify exits (both pre- and post-). eTrust AC verifies the password using its built-in features only when users replace their own passwords. All information relevant to the attempt to enter a new password is passed to the API function. This information includes the name of the user invoking the password utility; the name of the user whose password is being changed; the user's old password, if it exists; the user's new password; and the eTrust AC result. Results may be 0 (Okay) or 1 (Error). All the information is passed to the function in the structure SEOS_EXITPASS.

A *password change* event occurs whenever a user attempts to update an existing user password. All information relevant to the update attempt is passed to the API function. This information includes the name of the user invoking the password utility; the name of the user whose password is being changed; the user's new password; and both the eTrust AC and the system results. The information is passed to the function in the structure SEOS_EXITPASS.

Events Linked to pwdchange.dll in Windows

The *password quality check* and *password change* events are registered with the password dll pwdchange. After installing these events in the registry, you must reboot in order to have proper registration. The Exits API data structure SEOS_EXITPASS is used to pass information about these events between functions. For more information about the SEOS_EXITPASS data structure, see Structure and Data Types in this chapter.

User Information

(For UNIX only) eTrust AC passes as much information about the user requesting authorization as possible to the Exits API functions. Depending on what eTrust AC knows about the user, the Exits API function may be given the user name, the UNIX user ID number, and the user ACEE handle.

Users may or may not be defined in the eTrust AC database. If the user is defined in the database, eTrust AC has an entry in the Accessor Environment Entry (ACEE) table for that user. All entries in the ACEE table have an ACEE handle that points to the information about that entry. A user not defined in the database is assigned an ACEE handle of -1. An ACEE handle of -1 informs the Exits API functions that the request did not come from an eTrust AC-defined user.

Compiling and Linking

Compiling and Linking in UNIX with the api_authx Library

This section provides the instructions for compiling and linking your Exits API functions with the *seosd* daemon or the *sepass* utility. These are general instructions that describe the most common system configurations. Each system has its own specific requirements. It is impossible to provide detailed requirements for every possible system configuration. Consult your system guides for the details of your particular system's compiler and linker options.

You must include the header files *authxapi.h* and *seostype.h* in your Exits API functions. These files are located in the include subdirectory. Put the following two lines near the top of an Exits API function source file:

```
#include authxapi.h
#include seostype.h
```

We recommend using an ANSI-C compliant compiler.

Compiling and Linking the Callback Exits Dynamic Link Library in Windows

This section provides the instructions for compiling and linking your Exits API functions with the seosd service or the pwdchange.dll. These are general instructions that describe the most common system configurations. Each system has its own specific requirements. It is impossible to provide detailed requirements for every possible system configuration. Consult your system guides for the details of your particular system's compiler and linker options.

You must include the header file authxapi.h in your Exits API functions. This file is located in the include subdirectory. Put the following line near the top of an Exits API function source file:

```
#include authxapi.h
```

We recommend using an ANSI-C compliant compiler.

Linking Your Application with eTrust AC

Because the target of your code is either a shared library (for UNIX) or a dynamic link library (for Windows), you may need to use compiler flags to determine the correct code generation method. The examples provided in the samples subdirectory should help you find the appropriate flag for your operating system.

System Design and Limits

Modular Design

eTrust AC is completely modular in design and implementation. Management of resources is also completely modular. Most of the system objects that eTrust AC protects are considered general resources. A class is a family of resources that share the same attributes. For example, an attempt to open a file is considered an access request to a resource of class FILE. In UNIX, an attempt to substitute (su) to another user is considered an access request to a resource of class SURROGATE. Grouping resources in this manner allows eTrust AC to use one general authorization algorithm.

Your Exits API functions must maintain the same modular approach as the eTrust AC functions. The Exits API functions are called whenever there is an attempt to access a specified resource. Your functions must use a modular algorithm that works consistently for an entire class and does not interfere with or generate errors for other classes. For more information about classes, properties, and resources, see the *Administrator Guide* (for Windows or UNIX).

Important! eTrust AC constantly receives authorization requests from system events and user programs. These requests may be redirected to your exit function. Ensure that your function is optimized and terminates as quickly as possible, so as to minimize system overhead. Special care must be taken when writing exit functions. You cannot write an exit function and leave debugging for runtime. A trivial bug can bring down your system.

Configuration

For UNIX

After you compile your code, generate a shared library that contains the compiled version of your code. The apisamples directory contains sample functions and a makefile that demonstrate the process. Note that compilation for shared libraries usually requires additional compiler parameters to create position-independent code. Consult your compiler or linker documentation to learn how to create shared libraries in your particular system.

After you have written your code and created a shared library, add your shared library to the "on-demand" shared libraries configuration file relevant to the program to which your code should link.

If you have written a shared library for one of the following daemons or programs, add your shared library to the relevant file (you may need to create these .ext files using the examples in the apisamples directory):

- For selogrd, add your shared library to *eTrustACDir*/etc/selogrd.ext.
- For selogrcd, add your shared library to *eTrustACDir*/etc/selogrcd.ext.
- For seosd, add your shared library to *eTrustACDir*/etc/seosd.ext.
- For sepass, add your shared library to *eTrustACDir*/etc/sepass.ext.

Each file contains two columns: the driver name and the shared library path. By convention, the driver name is a string that has the same name as your target type; however, it can be any valid C language symbol.

For example, if you have written code to implement a pager, your target name should be *pager* and the complete file entry would be:

```
pager /usr/local/lib/libseospager.so
```

```
This file entry means that the daemon seosd loads the shared library at startup
and calls your function:
```

```
/usr/local/lib/libseospager.so
```

Although some systems support a predefined function called _init, we recommend that you use the functiondriver_RegisterDestination instead. This is really the first function called from the shared library. The function driver_RegisterDestination registers your new target type.

On daemon shutdown, we recommend that you use the function driver_UnregisterDestination instead of the predefined function _fini.

Note: Using the eTrust AC functions instead of the predefined system functions gives your code greater portability.

The daemon seosd uses the same file configuration format as sepass.

For Windows

After you compile your code, generate a dynamic link library that contains the compiled version of your code. The SDK directory contains sample functions and a makefile to demonstrate the process. Note that compilation for dynamic link libraries usually requires additional compiler parameters. Consult your compiler or linker documentation to learn how to create dynamic link libraries in your particular system.

After you have written your code and created a dynamic link library, you should install it in the eTrust AC registry tree as follows:

1. Create a new key with any name you choose under the following existing key:

   ```
   HKEY_LOCAL_MACHINE\Software\ComuterAssociates\eTrustccessControl\Exits\Engine
   ```

2. Under the newly created sub-key create two values:

   - A REG_SZ value with the name Path. The value should be the full path of your exits dynamic link library. This value is used by the seosd service and the pwdchange dll in loading the new exits dynamic link library.
   - A REG_SZ value with the name Prefix. The value should be any valid C language symbol, which will be the prefix of the registration function in your dynamic link library code. For example, if the value is “TEMP” then the registration function name in the dynamic link library should be TEMP_RegisterExit. This value is used by the seosd service and the pwdchange dll in loading the registration function address.

System Calls

For UNIX

Since eTrust AC intercepts operating system calls, not all system activities can be allowed while you are in the midst of an authorization action. The following functions cannot be called from an Exits API function of seosd:

In This Environment:	Do Not Use These Functions:

NIS or DNS servers running eTrust AC; Solaris 2.5.*x* and later	`getgrent`
	`getgrgid`
	`etgrnam`
	`gethostbyaddr`
	`gethostbyname`
	`gethostent`
	`getnetbyaddr`
	`getnetbyname`
	`getnetent`
	`getprotobyname`
	`getprotobynumber`
	`getprotoent`
	`getpwent`
	`getpwnam`
	`getpwuid`
	`getservbyname`
	`getservbyport`
	`getservent`

Any station running eTrust AC	`getrpcbyname`
	`getrpcbynumber`
	`getrpcent`

Error Codes

eTrust AC uses an error code defined as an integer, composed of two bytes. The MSB contains a layer code and the LSB contains an error code specific to that layer. This allows up to 256 different layers with 256 different error codes each. To simplify error code management, eTrust AC uses the macro _SEOS_RC. The Exits API also uses the macro AUTHXAPI_MODULE to define the layer code. Do not use these macros yourself in your code or you may have compilation problems.

Return Codes

eTrust AC uses the following convention for return codes: a return value of zero indicates success; any other value indicates an error.

Exits API Examples

The eTrust AC package includes the following two sample programs demonstrating use of the Exits API.

eTrust AC Daemon Exits for UNIX

The apisamples directory contains the API header files (in the include subdirectory) and the library functions (in the lib subdirectory).

The following program calls a special user-defined exit function upon any action by the seosd daemon. The function keeps statistics on the number of times each action is executed. When the function senses a login for user root, it simply prints the information gathered. A more detailed explanation of the main points of this function follows the source code.

```
/*===========================================================
Project  : eTrust
Module   : eTrust
Version  : 8.0
File     : seosdexits.c
Purpose  : eTrust daemon exits sample.
============================================================

Copyright :
Copyright 2004 Computer Associates International, Inc.
============================================================*/
#define __SEOSEXIT_C
/* System headers                       :
   *  sys/types.h                       : we cust using uid_t
   *  stdio.h                           : printf
   *  string.h & memory.h               : memcpy
 */
#include sys/types.h
#include stdio.h
#include string.h
#include memory.h

/* authx API header file */
#include authxapi.h

/* This a sample program for making an exit module
 *  for Access Control.
 * The module only 'printf' the information and
 * gathers statistics on the # of events.
 * When the 'root' user logs in the statistics
 * are printed.
 */

typedef struct
{   int nPreRes;
    int nPostRes;
    int nPreLogin;
    int nPostLogin;
    int nPreInet;
int nPostInet;
} EXIT_CALLS_COUNTERS;

static EXIT_CALLS_COUNTERS counters;

static void print_my_statistics(void)
{
    printf(“General Resource ... Pre %6d Post %6d\n”
           “Login ............. Pre %6d Post %6d\n”
           “Internet TCP ....... Pre %6d Post %6d\n”,
           counters.nPreRes, counters.nPostRes,
           counters.nPreLogin, counters.nPostLogin,
           counters.nPreInet, counters.nPostInet);
}

static int ExitFunc_PreResource(void *data, SEOS_EXITRES *p_sexr)
{
    SEOS_EXITGENR *ptr;

    ptr = (SEOS_EXITGENR *)data;
    counters.nPreRes++;
    printf(“Pre General Resource Class %s\n”, ptr->szClass);
    return 0;
}

static int ExitFunc_PostResource(void *data, SEOS_EXITRES *p_sexr)
{
    SEOS_EXITGENR *ptr;

    ptr = (SEOS_EXITGENR *)data;
    counters.nPostRes++;
    printf(“Post General Resource Class %s\n”, ptr->szClass);
    return 0;
}


static int ExitFunc_PreLogin(void *data, SEOS_EXITRES *p_sexr)
{
    SEOS_EXITLOGIN *p_sexl;
    char buff[20];
    char const *p;
    p_sexl = (SEOS_EXITLOGIN *)data;
    counters.nPreLogin++;
    if (p_sexl->szUname == NULL)

       {p = buff; sprintf(buff, “UID=%u”, p_sexl->luid);}
    else
        p = p_sexl->szUname;
    printf(“Pre Login For %s\n”, p);
    /* For ROOT print some statistics */
    if (p_sexl->luid == (uid_t)0)
        print_my_statistics();
    return 0;
}
static int ExitFunc_PostLogin(void *data, SEOS_EXITRES *p_sexr)
{
    SEOS_EXITLOGIN *p_sexl;
    char buff[20];
    char const *p;
    p_sexl = (SEOS_EXITLOGIN *)data;
    counters.nPostLogin++;
    if (p_sexl->szUname == NULL)
       {p = buff; sprintf(buff, “UID=%u”, p_sexl->luid);}
    else
        p = p_sexl->szUname;
    printf(“Post Login For %s\n”, p);
    /* For ROOT print some statistics */
    if (p_sexl->luid == (uid_t)0)
        print_my_statistics();
    return 0;
}
static int ExitFunc_PreInet(void *data, SEOS_EXITRES *p_sexr)
{
    /* Don't do too much work on TCP */
    counters.nPreInet++;
    return 0;
}
static int ExitFunc_PostInet(void *data, SEOS_EXITRES *p_sexr)
{
    /* Don't do too much work on TCP */
    counters.nPostInet++;
    return 0;
}
int sample_RegisterExit(void)
{
    int rc;
    memset(&counters, 0, sizeof(counters));
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_PREGNRES,
         ExitFunc_PreResource);
    if (rc) return rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_POSTGNRES,
         ExitFunc_PostResource);
    if (rc) return rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_PRELOGIN,
         ExitFunc_PreLogin);
    if (rc) return rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_POSTLOGIN,
         ExitFunc_PostLogin);
    if (rc) return rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_PREINET,
         ExitFunc_PreInet);
    if (rc) return rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_POSTINET,
         ExitFunc_PostInet);
    return rc;
}
void sample_UnregisterExit(void)
{
    /* We don't have anything to do in this example */
}
```

The authxapi.h header file contains the prototypes and definitions required to use the API. The code declares a new type EXIT_CALLS_COUNTER. This structure consists of counters for each event the API registers. A static variable of this new local type is declared. The general resource check and TCP/IP request functions listed are simple counters. The login functions have the same counters and also compare the user ID of the logged-in user to 0. The login functions take advantage of the fact that the UNIX user ID for root is 0 to avoid fetching the real user information from system databases.

Note: This example prints to the screen. eTrust AC daemons should not use screen output as it causes a significant decline in performance.

eTrust AC SeOS Engine Exits for Windows

The SDK directory contains the API header files (in the include subdirectory) and the library functions (in the lib subdirectory).

The following program calls a special user-defined exit function upon any action by the seosd service. The function keeps statistics on the number of times each action is executed. When the function senses a login for user Administrator, it simply prints the information gathered. A more detailed explanation of the main points of this function follows the source code.

```
/*=========================================================================
   Project:    eTrust Access Control 8.0 for Windows NT/2000
   Module:     eTrust Engine Exits sample.
   File:       main.c
   Purpose:    Sample for eTrust Engine Exits DLL to demonstrate how user
               can use eTrust Engine authorization in Pre and Post checks.
   Usage:      Build ExitsExample.dll and use eTrust Access Control 8.0 for
               Windows NT/2000 documentation to config your registry and file
               system.
   ========================================================================
     Copyright 2004 Computer Associates International, Inc.
   ======================================================================== */

#include <stdio.h>
#include <authxapi.h>

typedef struct
{
     int nPreRes;
     int nPostRes;
     int nPreLogin;
     int nPostLogin;
} EXIT_CALLS_COUNTERS;

EXIT_CALLS_COUNTERS counters = { 0 };

static void print_my_statistics(void)
{
     //
     // print statistic check counters
     //
     printf("General Resource ... Pre %6d Post %6d \n"
             "Login...............Pre %6d Post %6d \n",
             counters.nPreRes, counters.nPostRes,
             counters.nPreLogin, counters.nPostLogin);
}

static int ExitFunc_PreResource(void* data, SEOS_EXITRES *p_sexr)
{
     SEOS_EXITGENR *ptr;

     ptr = (SEOS_EXITGENR *) data;

     counters.nPreRes++;

     //
     // Pre check: Print eTrust Class name of the resource
     //
     printf("Pre General Resource %s of Class %s\n", ptr->szRes, ptr->szClass);
     return 0;
}

static int ExitFunc_PostResource(void* data, SEOS_EXITRES *p_sexr)
{
     SEOS_EXITGENR *ptr;

     ptr = (SEOS_EXITGENR *) data;
     counters.nPostRes++;

     //
     // Post check: Print eTrust Class name of the resource
     //

     printf("Post General Resource %s of Class %s\n", ptr->szRes, ptr->szClass);

     return 0;
}

static int ExitFunc_PreLogin(void* data, SEOS_EXITRES *p_sexr)
{
     SEOS_EXITLOGIN *p_sexl;
     char buff[20];
     char const *p;

     p_sexl = (SEOS_EXITLOGIN*) data;
     counters.nPreLogin++;
     if(p_sexl->szUname == NULL)
     {
          p = buff;
          sprintf(buff, "UID = %u", p_sexl->luid);
     }
     else
          p = p_sexl->szUname;

     //
     // For Pre Login check: print the Login User name
     //
     printf("Pre Login For %s from terminal %s\n", p, p_sexl->szTerm);

     if(strstr(p_sexl->szUname, "Administrator") != NULL)
     {
          print_my_statistics();
     }

     return 0;
}

static int ExitFunc_PostLogin(void* data, SEOS_EXITRES *p_sexr)
```

```
{
     SEOS_EXITLOGIN *p_sexl;
     char buff[20];
     char const *p;

     p_sexl = (SEOS_EXITLOGIN*) data;
     counters.nPostLogin++;

     if(p_sexl->szUname == NULL)
     {
          p = buff;
          sprintf(buff, "UID = %u", p_sexl->luid);
     }
     else
          p = p_sexl->szUname;

     //
     // For Post Login check: print the Login User name
     //
     printf("Post Login For %s from terminal %s\n", p, p_sexl->szTerm);

     if(strstr(p_sexl->szUname, "Administrator") != NULL)
     {
          print_my_statistics();
     }

     return 0;
}

typedef int(*PFNEXIT)(void);


// Function name: SM_RegisterExit
// Description: Export exits function
// Return type: int 0
// Argument: IN/OUT PFNEXIT pfnExit[] - array of user defined functions
// Argument: IN/OUT DWORD *pdwType - array of Engine events
_declspec(dllexport) int __stdcall SM_RegisterExit(PFNEXIT pfnExit[], PDWORD
pdwType)
{
     pfnExit[0] = (PFNEXIT)ExitFunc_PreLogin;
     pfnExit[1] = (PFNEXIT)ExitFunc_PostLogin;
     pfnExit[2] = (PFNEXIT)ExitFunc_PreResource;
     pfnExit[3] = (PFNEXIT)ExitFunc_PostResource;
     pdwType[0] = AUTHXAPI_EV_PRELOGIN;
     pdwType[1] = AUTHXAPI_EV_POSTLOGIN;
     pdwType[2] = AUTHXAPI_EV_PREGNRES;
     pdwType[3] = AUTHXAPI_EV_POSTGNRES;
```

```
    return 0;
}
```

Additionally you will need another .def file in the project that contains:

- LIBRARY-ExitsExample
- DESCRIPTION-ExitsExample Windows Dynamic Link Library
- EXPORTS-SM_RegisterExit

The authxapi.h header file contains the prototypes and definitions required to use the API. The code declares a new type EXIT_CALLS_COUNTER. This structure consists of counters for each event the API registers. A static variable of this new local type is declared. The general resource check functions listed are simple counters. The login functions have the same counters and also compare the user name of the logged-in user to Administrator.

Note: This example prints to the screen. eTrust AC services should not use screen output because it is not a console application.

Password Utilities Exits

For UNIX

The following code demonstrates a simple user exit function for password verification. This code compares the potential password to the string password and denies permission to choose the password if a match is found.

```
/*===========================================================
Project    : eTrust
Module     : eTrust
Version    : 8.0
File       : exit.c
Purpose    : Example of exit function for passwords utilities
============================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
============================================================*/
/* Example for using password library to extend password rules*/
/* This function verifies the password before Access Control */
/* does. To override Access Control checks, use result */
/* SEOS_EXITR_PASS.      */
/* Look at Makefile.exits for compilation options. */

#include API/authxapi.h

/* This function does not allow user to use the word “password”*/
/* as the new password.                                       */
int ExitFunc_PreVerify(void *sexp, SEOS_EXITRES *sexr)
{
    if (strcmp(((SEOS_EXITPASS *) sexp)->szPass,”password”) == 0)
    {
       printf(“new password is refused by exit function\n”);
       sexr->result = SEOS_EXITR_DENY;
                  /* Do not allow that password */
    }
    else
       sexr->result = SEOS_EXITR_CHECK;
                  /* Continue with Access Control checks  */
    return 0;
}
/* must register the above exit function */
int sample_RegisterExit(void)
{
    int rc;
    rc = authxapi_RegisterExitFunction(AUTHXAPI_EV_PREVERPWD,
             ExitFunc_PreVerify);
    if (rc) return rc;
}
void sample_UnregisterExit(void)
```

```
{
    /* Nothing to do really in this case */
}
```

For Windows

The following code demonstrates a simple user exit function for password verification. This code compares the potential password to the string password and denies permission to choose the password if a match is found.

```
/* ==========================================================================
     Project: eTrust Access Control 8.0 for Windows NT/2000
     Module: eTrust Password Exits sample.
     File: main.c
     Purpose: Sample for eTrust Password Exits DLL to demonstrate how user
an
              use eTrust Password authorization in Pre and Post checks.
     Usage: Build PasswordExitsExample.dll and use eTrust Access Control
.0 for
            Windows NT/2000 documentation to config your registry and file
            system.

=======================================================================
     Copyright 2004 Computer Associates International, Inc.

======================================================================= */

#include <stdio.h>
#include <authxapi.h>

static int ExitFunc_PreVerifyPassword(void* data, SEOS_EXITRES *p_sexr)
{
     SEOS_EXITPASS *ptr;

     ptr = (SEOS_EXITPASS *) data;

     if (strcmp(ptr->szPass, "password") == 0)
     {
          p_sexr->result = SEOS_EXITR_DENY;
          /* Do not allow that password */
     }
     else
     {
          /* Continue with Access Control checks */
          p_sexr->result = SEOS_EXITR_CHECK;
     }

     return 0;
}

typedef int(*PFNEXIT)(void);


// Function name: PWD_RegisterExit
// Description: Export exits function
// Return type: int 0
// Argument: IN/OUT PFNEXIT pfnExit[] - array of user defined functions
// Argument: IN/OUT DWORD *pdwType - array of Engine events
_declspec(dllexport) int __stdcall PWD_RegisterExit(PFNEXIT pfnExit[],
DWORD pdwType)
{
     pfnExit[0] = (PFNEXIT)ExitFunc_PreVerifyPassword;
     pdwType[0] = AUTHXAPI_EV_PREVERPWD;
     return 0;
}
```

Note: In order to use password-related exits, after installing the exits dynamic link library in the registry, you must reboot the machine. This is mandatory because the pwdchange dynamic link library that handles eTrust AC password verification is loaded only when the machine starts up.

Exits API Functions for UNIX

The Exits API functions provided by eTrust AC are grouped according to the following categories:

- General functions are used by all Exits API functions, whether they are linked to the seosd daemon or the password utility sepass. The tasks performed by these functions include registration, removal of registration, initialization, and termination.
- Database interface functions are applicable only to Exits API functions linked to seosd.
- Shared library functions are functions provided by eTrust AC.

General Functions

authxapi_IsThereExitFunction

Checks if an Exits API function for a specific event has been registered with eTrust AC.

authxapi_RegisterExitFunction

Registers your new Exits API function with eTrust AC.

authxapi_UnregisterExitFunction

Removes an Exits API function that was previously registered with eTrust AC.

Database Interface Functions

authxapi_FreeListValues

Frees the memory allocated during a previous call to the function authxapi_GetObjectListValue.

authxapi_GetObjectListValue

Retrieves the values of a list value property from a database object.

authxapi_GetObjectProperty

Retrieves the value of a single value property from a database object.

authxapi_GetUserInfo

Retrieves user name when given a user handle from an Exits API function.

Shared Library Functions

driver_RegisterExit

Provided by the extension and called on program startup.

driver_UnregisterExit

Provided by the extension and called on program termination.

authxapi_RegisterExitFunction Function

Valid on UNIX only.

The authxapi_RegisterExitFunction function registers an exit function for a specific event. Registration should be handled during program startup and shutdown, although it can be performed at any stage.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno and returns one of the following error codes:

Return Value	ERRNO	Meaning
AUTHXAPI_E_OCCUPIED	EEXIST	Event already registered.
AUTHXAPI_E_NOEVENT	EINVAL	Invalid event code.

```
int authxapi_RegisterExitFunction (int event, PFSeosExitFunc user_func);
```

event

Code of the event to which your Exits API function is registered. Password events are available only when linked to the password utility. General system events are available only when linked to the seosd daemon.

Valid **event** values are:

AUTHXAPI_EV_PRELOGIN

Pre-login event

AUTHXAPI_EV_POSTLOGIN

Post-login event

AUTHXAPI_EV_PREGNRES

Pre-general resource event

AUTHXAPI_EV_POSTGNRES

Post-general resource event

AUTHXAPI_EV_PREINET

Pre-TCP/IP request event

AUTHXAPI_EV_POSTINET

Post-TCP/IP request event

AUTHXAPI_EV_PREVERPWD

Pre-password quality check event

AUTHXAPI_EV_POSTVERPWD

Post-password quality check event

AUTHXAPI_EV_PRESETPWD

Pre-password change event

AUTHXAPI_EV_POSTSETPWD

Post-password change event

user_func

Pointer to the user function that eTrust AC should call when the specified event occurs.

Example: Registering a User's Exit Function.

This example illustrates how you use the authxapi_RegisterExitFunction function to deny all login attempts for the user *ismith*.

```
/* Sample function to deny all login attempts of user 'jsmith' */
int MyExitFunc(void *exit_data, SEOS_EXITRES *res)
```

```
{
    SEOS_EXITLOGIN *login_data;

    login_data = (SEOS_EXITLOGIN *)exit_data;
    if (login_data->szUname != NULL)
    {
             if (strcmp(login_data->szUname, “jsmith”))
                 return 0;
             res->result = SEOS_EXITR_DENY;
    }
    return 0;
}

/* The function Access Control looks for on initialization */
/* of extension. */
int sample_RegisterExit(void)
{
    return authxapi_RegisterExitFunction(AUTHXAPI_EV_PRELOGIN,
                                         MyExitFunc);
}
```

More information:

authxapi_UnregisterExitFunction Function (see page 80)

authxapi_UnregisterExitFunction Function

Valid on UNIX only.

The authxapi_UnregisterExitFunction function unregisters your exit function from eTrust AC.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno to EINVAL and returns AUTHXAPI_E_NOEVENT, meaning that the event code passed was invalid.

```
int authxapi_UnregisterExitFunction (int event);
```

event

Integer code of the event (see page 77) to which your Exits API function is registered.

Example: Unregistering User's Exit Function

This example illustrates how you use the authxapi_UnregisterExitFunction to unregister a user's exit function.

```
rc = authxapi_UnregisterExitFunction(AUTHXAPI_EV_PRELOGIN);
if (rc)
{
     syslog(LOG_ERR,
     “Unexpected error unregistering exit function [%m]”);
}
```

More information:

authxapi_RegisterExitFunction Function (see page 77)

authxapi_IsThereExitFunction Function

Valid on UNIX only.

The authxapi_IsThereExitFunction function determines whether an exit function has been registered for a specific event.

If the exit function exists, this function returns TRUE (1). If it does not exist, this function returns FALSE (0).

```
int authxapi_IsThereExitFunction (int event);
```

event

Integer code of the event (see page 77) being checked.

Example: Checking If Exit Function Exists.

This example illustrates how you use the authxapi_IsThereExitFunction function to check if there is a pre-login exit function.

```
{
    int rc;
    ...
    rc = authxapi_IsThereExitFunction(AUTHXAPI_EV_PRELOGIN);
    if (rc)
            printf("There is a pre-login exit function\n");
    else
            printf("There is no pre-login exit function.\n");
}
```

More information:

authxapi_RegisterExitFunction Function (see page 77)
authxapi_UnregisterExitFunction Function (see page 80)

authxapi_GetObjectProperty Function

Valid on UNIX only.

The authxapi_GetObjectProperty function retrieves the value of a single-value property of an object stored in the database. Properties that have multiple values, such as lists, cannot be retrieved with this function. You retrieve property lists with the authxapi_GetObjectListValue function. All parameter strings must be NULL terminated.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno and returns one of the following error codes:

Return Value	ERRNO	Meaning
AUTHXAPI_E_EINVAL	EINVAL	Invalid (NULL) pointers
AUTHXAPI_E_INVOBJ	EINVAL	Invalid object descriptor
AUTHXAPI_E_INVPROP	EINVAL	Invalid property descriptor
AUTHXAPI_E_NOCLASS	ENOENT	Required class not found
AUTHXAPI_E_NOOBJ	ENOENT	Required object not found
AUTHXAPI_E_NOPROP	ENOENT	Required property not found
AUTHXAPI_E_PTYPE	EINVAL	Property type is a list
AUTHXAPI_E_DBERROR	EIO	Suspect corruption of database
AUTHXAPI_E_NOVAL	ENOENT	No value for property associated with this object

```
int authxapi_GetObjectProperty (const char  *szClass,
                                const char  *szObj,
                                SEOSDB_ODF  *p_odf,
                                const char  *sz_Prop,
                                SEOSDB_PDF  *p_pdf,
                                void        *val,
                                int         *size);
```

szClass

The name of the class to which the resource belongs.

szObj

The name of the object (record) whose property value you want to retrieve.

p_odf

Points to an object descriptor fetched by this function or provided by the caller from a previous call to an Exits API get function.

szProp

The name of the property whose value you want to retrieve.

p_pdf

A pointer to a property descriptor fetched by this function or provided by the caller from a previous call to an Exits API get function.

val

A pointer to a variable that is filled with the value of the property. The caller should provide a pointer to a variable that is of the same type as the property's data type.

size

A pointer to the size, in bytes, of the region in memory to which the val parameter is pointing.

Example: Fetching a Value from the Database.

This example, part of an exit function for a general resource, retrieves a value stored in the database. The exit function checks that the request is for the correct resource class. Other classes are ignored. The authxapi_GetObjectProperty function is called to retrieve the value for the COMMENT property. If the retrieval was successful, the function tests if the string contains the substring “ NO ” (note the spaces before and after the word). If so, the request is denied. Otherwise the function returns control to eTrust AC to perform its standard checks.

```
#include stdio.h
#include authxapi.h
#include seostype.h
```

```
int MyExitFunction(void *exit_data, SEOS_EXITRES *result)
{
    SEOS_EXITGENR *genr_data;
    SEOSDB_ODF odf;
    SEOSDB_PDF pdf;
    SEOS_COMMENT comment;
    int rc;

    genr_data = (SEOS_EXITGENR *)exit_data;
    /* Ignore any class that is not of interest */
    if (strcmp(genr_data->szClass, “MyClass”))
        return 0;

    /* Fetch the information for the COMMENT property of the */
    /* resource to which access is verified. */
    rc = authxapi_GetObjectProperty(“MyClass”, genr_data->szRes,
               &odf, “COMMENT”, &pdf, comment, sizeof(comment));

    if (rc == 0)
    {
        /* We have now the comment string. Does it contain “NO”?*/
        if (strstr(comment, “ NO “) != NULL)
        {
            result->result = SEOS_EXITR_DENY;
        }
    }
    return 0;
}
```

Note: To update the COMMENT field from the selang command interpreter, you should use the comment argument. For example:

```
newres MyClass anobject \
comment('This object has “ NO “ in the comment property')
```

authxapi_GetObjectListValue Function

Valid on UNIX only.

The authxapi_GetObjectListValue function retrieves a list of values assigned to a property of an object stored in the database. Properties that have single values cannot be retrieved with this function. You retrieve single value properties with authxapi_GetObjectProperty.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno and returns one of the following error codes:

Return Value	ERRNO	Meaning
AUTHXAPI_E_EINVAL	EINVAL	Invalid (NULL) pointers
AUTHXAPI_E_INVOBJ	EINVAL	Invalid object descriptor
AUTHXAPI_E_INVPROP	EINVAL	Invalid property descriptor
AUTHXAPI_E_NOCLASS	ENOENT	Required class not found
AUTHXAPI_E_NOOBJ	ENOENT	Required object not found
AUTHXAPI_E_NOPROP	ENOENT	Required property not found
AUTHXAPI_E_PTYPE	EINVAL	Property type is a list
AUTHXAPI_E_DBERROR	EIO	Suspect corruption of database
AUTHXAPI_E_NOVAL	ENOENT	No value for property associated with this object

```
int authxapi_GetObjectListValue (const char   *szClass,
                                 const char   *szObj,
                                 SEOSDB_ODF   *p_odf,
                                 const char   *szProp,
                                 SEOSDB_PDF   *p_pdf,
                                 void         ***val,
                                 unsigned int *psize,
                                 unsigned int *count);
```

szClass

The name of the class to which the resource belongs.

szObj

The name of the object whose property value you want to retrieve.

p_odf

A pointer to an object descriptor fetched by this function or provided by the caller from a previous call to an Exits API get function.

szProp

The name of the property whose values you want to retrieve.

p_pdf

A pointer to a property descriptor fetched by this function or provided by the caller from a previous call to an Exits API get function.

val

A pointer to a variable that is assigned the memory address of an array of pointers. The array of pointers point to the data values being retrieved. The authxapi_GetObjectListValue function allocates the memory used here.

psize

Size in bytes of the region in memory allocated to each element in the value list.

count

The number of elements in the value list. May be 0 if no elements are found.

Notes:

- If the szObj parameter is NULL, the function assumes p_odf is pointing to a valid object descriptor obtained from a previous call to one of the get functions provided by the Exits API. This speeds up processing when you are dealing with a series of objects that share the same object descriptor, since you do not spend time repeatedly fetching the same object descriptor.

Although it is faster to use an object descriptor than an object name, it is not safe to store the object descriptors in global variables and to use them in subsequent calls to this function. This is because updates to the database may delete these objects.

- If the szProp parameter is NULL, the function assumes p_pdf is pointing to a valid property descriptor obtained from a previous call to one of the get functions provided by the Exits API. This speeds up processing when you are dealing with a series of objects that share the same property descriptor, since you do not spend time repeatedly fetching the same property descriptor.

  It is safe to store property descriptors in global variables and use them in subsequent calls to this function, as property definitions are not subject to change while the seosd daemon is active.

The authxapi_GetObjectListValue function allocates a vector of void pointers, each pointing to an allocated buffer that holds a single element in the list of values. You must declare a list variable of any type as a pointer to a pointer, such as int **. A pointer to this list variable is then passed into authxapi_GetObjectListValue, typecast as a (void ***). For example:

```
{
    int **list;
    unsigned int psize, count;
    int rc;
    ...
rc = authxapi_GetObjectListValue(szClass,
         szObj, &odf, szProp, &pdf,
         (void ***)&list, &psize, &count);
...
}
```

When authxapi_GetObjectListValue returns, the list variable points to a newly allocated area of memory containing the pointers, stored sequentially from 0 to count, pointing to each list item. Each list item is stored in yet another newly allocated memory area. Be sure to use authxapi_FreeListValues to free all the memory allocated by authxapi_GetObjectListValue.

For example, when you have a list of N data elements, your memory is allocated as follows:

Argument	**Description**
[elem0]	1st data element
[elem1]	2nd data element
[elem*N*]	N+1 data element

Example: Retrieve List of Values of a Property Resource

The following Exits API function retrieves a list of values from a list-type property of a resource from the database, and then loops through the list of values. The code that actually uses the information retrieved is not shown.

```
#include stdio.h
#include authxapi.h
#include seostype.h

int MyExitFunction(void *exit_data, SEOS_EXITRES *result)
{
    SEOS_EXITGENR *genr_data;
    SEOSDB_ODF odf;
    SEOSDB_PDF pdf;
    SEOS_ACL **access_list, *acl_element;
    int rc;
    unsigned int psize, count, counter;
    genr_data = (SEOS_EXITGENR *)exit_data;
    /* Ignore any class that is not of interest */
    if (strcmp(genr_data->szClass, “MyResClass”))
        return 0;
    /* Fetch the information for the ACL property of the */
    /* resource being accessed. */
    rc = authxapi_GetObjectListValue(“MyResClass”, genr_data->szRes
           &odf, “ACL”, &pdf, (void ***)access_list, &psize, &count);
    if (rc == 0)
    {
         /* We have the ACL now. Lets just see a demonstration */
         /* of looping through the list. */
         for (counter = 0; counter < count; counter++)
         {
               acl_element = access_list[counter];
               /*
                * > > > User Code Here < < <
                */
         }
         authxapi_FreeListValues((void ***) &access_list, &count);
    }
    return 0;
}
```

More information:

authxapi_FreeListValues (see page 89)
authxapi_GetObjectProperty Function (see page 82)

authxapi_FreeListValues

Valid on UNIX only.

The authxapi_FreeListValues function frees the memory allocated when a list of values was retrieved during a previous call to the authxapi_GetObjectListValue function.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno to EINVAL and returns an error code AUTHXAPI_E_EINVAL, indicating that one of the pointers passed in was NULL.

```
int authxapi_FreeListValues (void ***value, unsigned int *count);
```

value

A pointer to the list of values held in the memory. Set to NULL when memory is successfully freed.

count

Number of elements in the value list. May be 0 if no elements are found. Set to 0 when memory is successfully freed.

More information:

authxapi_GetObjectListValue Function (see page 85)

authxapi_GetUserInfo Function

Valid for UNIX only.

The authxapi_GetUserInfo function retrieves a user name when given an eTrust AC user's ACEE handle.

If the function succeeds, it returns 0. If it fails, it sets the global variable errno and returns one of the following error codes:

Return Value	ERRNO	Meaning
AUTHXAPI_E_EINVAL	EINVAL	Invalid (NULL) pointers
AUTHXAPI_E_INVOBJ	EINVAL	Invalid object descriptor
AUTHXAPI_E_INVPROP	EINVAL	Invalid property descriptor
AUTHXAPI_E_NOCLASS	ENOENT	Required class not found
AUTHXAPI_E_NOOBJ	ENOENT	Required object not found

Return Value	ERRNO	Meaning
AUTHXAPI_E_NOPROP	ENOENT	Required property not found
AUTHXAPI_E_PTYPE	EINVAL	Property type is a list
AUTHXAPI_E_DBERROR	EIO	Suspect corruption of database
AUTHXAPI_E_NOVAL	ENOENT	No value for property associated with this object

```
int authxapi_GetUserInfo (int  seos_handle,
                          char *uname,
                          int  *size);
```

seos_handle

The handle of the user whose user name you are requesting.

uname

Buffer large enough to contain the user name being returned. Some UNIX systems allow no more than 8 characters per name; others allow more. eTrust AC treats users as any other object and allows up to 255 bytes per name.

size

On entry, the size in bytes of the memory area pointed to by uname. On return, the length of the user name string.

Example: Getting User Name String

This example illustrates how you get the user name string.

```
{
    int rc;
    char name[256];
    int size;
    ...
    size = sizeof (name)
    rc = authxapi_GetUserInfo(seos_handle, name, &size);
        ...
}
```

Exits API Functions for Windows

The API exit functions provided by eTrust AC are actually classified as callbacks. They are grouped into two categories:

- User-defined Functions-User-defined functions are called during eTrust AC execution, each function according to its registered type.
- Registration and Initialization Functions-This function is called once during seosd service and pwdchange.dll start up. Its purpose is to initialize any internal data and to register user-defined functions.

UserDefinedFunction Function

Valid for Windows only.

The UserDefinedFunction function is registered with a specific event type when the seosd service and the pwdchange.dll start. It is called according to the event type.

If the function succeeds, it returns 0. Any other numbers indicate a failure; the execution continues as usual-as if no callback function was called.

```
int UserDefinedFunction(void* data, SEOS_EXITRES * pExitResult);
```

data

A structure corresponding to the event type with which this function was registered. The data is cast into the following structures:

- SEOS_EXITGENR-For the following event types: AUTHXAPI_EV_PREGNRES, AUTHXAPI_EV_POSTGNRES.
- SEOS_EXITLOGIN-For the following event types: AUTHXAPI_EV_PRELOGIN, AUTHXAPI_EV_POSTLOGIN.
- SEOS_EXITPASS-For the following event types: AUTHXAPI_EV_PREVERPWD, AUTHXAPI_EV_POSTVERPWD, AUTHXAPI_EV_PRESETPWD, AUTHXAPI_EV_POSTSETPWD.

pExitResult

A pointer for the result structure which the exit function fills with information.

Example

Defining a user's exit function.

```
/* Sample function to deny all login attempts of user 'jsmith' */
int MyExitFunc(void *exit_data, SEOS_EXITRES *res)
{
     SEOS_EXITLOGIN *login_data;
     login_data = (SEOS_EXITLOGIN *)exit_data;
If (login_data->szUname != NULL)
     {
          if (strcmp(login_data->szUname, “jsmith”))
               return 0;
          res->result = SEOS_EXITR_DENY;
     }
     return 0;
}
```

UserDefinedPrefix_RegisterExit Function

Valid for Windows only.

The *UserDefinedPrefix*_RegisterExit function registers exit functions for some events. The registration in seosd service and pwdchange.dll is handled during startup.

If the function succeeds, it returns 0. Any other numbers indicate a failure; the execution continues as usual-as if no callback function was called.

Note: The *UserDefinedPrefix* is read from the registry (see Configuration requirements in this chapter) by the seosd service and pwdchange.dll prior to calling this function.

```
declspec(dllexport) int __stdcall UserDefinedPrefix_RegisterExit(PFNEXIT
pFunctionsExit[], unsigned long* pEventsType).
```

where PFNEXIT is defined as a pointer to a function that receives void as an argument and returns int.

```
int UserDefinedFunction(void* data, SEOS_EXITRES * pExitResult);
```

pFunctionsExit

An array of user-defined functions that are to be registered in the seosd service and in pwdchange.dll.

pEventsType

An array of event types corresponding to the function in the same entry in the pFunctionsExit argument.

Valid event types are:

- AUTHXAPI_EV_PRELOGIN-Pre-login event
- AUTHXAPI_EV_POSTLOGIN-Post-login event
- AUTHXAPI_EV_PREGNRES-Pre-general resource event
- AUTHXAPI_EV_POSTGNRES-Post-general resource event
- AUTHXAPI_EV_PREVERPWD-Pre-password quality check event
- AUTHXAPI_EV_POSTVERPWD-Post-password quality check event
- AUTHXAPI_EV_PRESETPWD-Pre-password change event
- AUTHXAPI_EV_POSTSETPWD-Post-password change event

Example

Registering a user's exit function.

```
/* Sample of registering a user defined function that will be called prior
```

```
to eTrust AC password verification. (The user has configured in the registry the
prefix PWD */
typedef int(*PFNEXIT)(void);
declspec(dllexport) int __stdcall PWD_RegisterExit(PFNEXIT
pFunctionExit[],unsigned long* pEventTypes)
{
     pFunctionExit[0] = (PFNEXIT)MyExitFunc;
     pEventTypes[0] = AUTHXAPI_EV_PREVERPWD;
     return 0;
}
```

Structure and Data Types

This section describes the data structures used by Exits API functions to pass information back and forth between the functions and the eTrust AC daemons/services. Which structure you use depends on the event being intercepted by your exit function.

The following describes each field of these data structures:

SEOS_EXITLOGIN

Data structure for login events

SEOS_EXITGENR

Data structure for general resource check events

SEOS_EXITINET

Data structure for TCP/IP request events (UNIX only)

SEOS_EXITPASS

Data structure for password quality check and password change events

SEOS_EXITRES

Data structure for results being returned to eTrust AC after any event

SEOSDB_ODF

Data structure for definition of an object in the database (UNIX only)

SEOSDB_PDF

Data structure for definition of a property in the database (UNIX only)

PFSeosExitFunc

Pointer to a function

SEOS_ACCESS

Encapsulates a single member of type SEOS_ACCS

SEOS_ACCS

Holds a list of access flags

SEOS_CID

Contains the class identification descriptor (UNIX only)

SEOS_OID

Contains the object identification descriptor (UNIX only)

SEOS_PID

Contains the property identification descriptor (UNIX only)

SEOS_EXITLOGIN

The first parameter passed to exit functions linked to attempted login events is a pointer to the SEOS_EXITLOGIN structure. This structure contains information about the attempted login. The SEOS_EXITLOGIN structure can be found in the *authxapi.h* file.

uid_t luid

User ID of the user trying to log in.

char const *szUname

Name of the user trying to log in.

char const *szTerm

Name of the terminal from which the user is trying to log in. Set to NULL when eTrust AC starts up.

dev_t device

Device number of the program trying to log in.

ino_t inode

Inode number of the program trying to log in.

char const *szProg

Name of the program trying to log in. Set to NULL when not applicable.

SEOS_EXITGENR

The first parameter passed to exit functions linked to attempted general resource check events is a pointer to the SEOS_EXITGENR structure. This structure contains information about the user and resource being verified. The SEOS_EXITGENR structure can be found in the *authxapi.h* file.

char const *szClass

Name of the general resource class being accessed. Check only the classes you explicitly decide to verify, and ignore the others.

char const *szRes

Name of the resource being accessed.

uid_t uid

The UNIX or Windows user ID of the user attempting access. Set to -1 when not applicable.

int seos_handle

ACEE handle associated with the user attempting access. Negative if the user is not defined in eTrust AC; 0 or positive otherwise.

char const *szUserName

Name of the user attempting access.

dev_t device

Device number of the program attempting access. Set to 0 when not applicable.

ino_t inode

Inode number of the program attempting access. Set to 0 when not applicable.

char const *szTerm

Name of the terminal from which user is attempting access. If user is not at a local terminal, this is set to the remote host name.

SEOS_ACCESS access_info

An unsigned long integer representing the type of access requested. For a list of possible values, see the SEOS_ACCS data type in the chapter “Authorization and Authentication API.”

char const *szProg

Name of the program attempting access. Set to NULL when not applicable.

SEOS_EXITINET

Valid for UNIX only. The first parameter passed to exit functions linked to attempted TCP/IP request events is a pointer to the SEOS_EXITINET structure. This structure contains information about the connection being requested. The SEOS_EXITINET structure can be found in the authxapi.h file.

char const *ClientAddr

IP address of the host requesting the connection.

char const *szHostName

Name of the host requesting the connection.

int Port

Number of the port to which connection is requested.

int Protocol

Protocol code used for the connection request. Currently, only TCP is supported.

SEOS_ACCESS accs_info

The exact level of connection access requested. For a list of possible values, see the SEOS_ACCS data type in the chapter “Authorization and Authentication API.” Currently, only read access is available for TCP/IP requests.

char const *szProg

Name of the program requesting a connection. Set to NULL when not applicable.

SEOS_EXITPASS

The first parameter passed to exit functions linked to attempted password quality check events is a pointer to the SEOS_EXITPASS structure. This structure contains information about the password being validated. The SEOS_EXITPASS structure can be found in the authxapi.h file.

char const *szIname

Name of the user starting the program to validate or set the password. This can be the user or an administrator (such as root, in UNIX).

char const *szUname

Name of the user whose password is being validated.

char const *szPass

New user password in clear text.

char const *szOldPass

Old user password. Defined only when users without the ADMIN attribute are changing their own current passwords. Set to NULL when undefined, such as when the administrator (root, in UNIX) is modifying another user's password.

int se_result

Result of the eTrust AC password verification mechanism. This field is not defined in pre-verification functions. In post-verification functions, this field holds the result of the eTrust AC password quality check. When used with the post set exit function, se_result holds a mask containing one of the following integer values:

0 VERIFYPASS_OK

Password is OK

1 VERIFYPASS_LEN

Password is too short

2 VERIFYPASS_NAME

Password contains the user's name

3 VERIFYPASS_MINS

Password contains too few lowercase characters

4 VERIFYPASS_MINC

Password contains too few uppercase characters

5 VERIFYPASS_MINN

Password contains too few numeric characters arguments

6 VERIFYPASS_MINO

Password contains too few special characters

7 VERIFYPASS_REP

Password contains too many repetitions of the same character

8 VERIFYPASS_SAME

New password is the same as the old one

9 VERIFYPASS_ASOLD

New password is the same as one of the values stored in the password history list

10 VERIFYPASS_ALFA

Password contains too few alphabetic characters

11 VERIFYPASS_ALFAN

Password contains too few alphanumeric characters

12 VERIFYPASS_TIME

Not enough time has passed since the last time the password was changed

13 VERIFYPASS_PREVCONTAIN

The old password is contained in the new one or vice versa

100 VERIFYPASS_BADARGB

The old password is bad

sys_result

Result of eTrust AC password-setting mechanism. Not defined in password pre-setting function. In post-setting function, this field holds the result of the eTrust AC attempt to change the password. This parameter is not currently used.

SEOS_EXITRES

Each Exits API function is passed a pointer to the SEOS_EXITRES structure as its second parameter. Pre-exit functions receive an empty structure that the functions fill with their results before returning control to the seosd daemon in UNIX or seosd service in Windows. Post-exit functions receive a structure filled with the results of the eTrust AC authorization. The functions then refill the structure with their own results before returning control to the seosd daemon in UNIX or seosd service in Windows. The SEOS_EXITRES structure can be found in the authxapi.h file.

int result

Final result of this exit function. Valid values are one of the following functions:

SEOS_EXITR_PASS

Instructs eTrust AC to permit the request.

SEOS_EXITR_DENY

Instructs eTrust AC to deny the request.

SEOS_EXITR_CHECK

Instructs eTrust AC to make the decision.

int stage

Stage at which the authorization process made the decision to grant or deny the request. These stages are listed in the *Reference Guide* (for Windows or UNIX), and the UNIX *Utilities Guide*, as well as in the header file seauthstages.h.

int gstage

Stage at which the authorization process was granted. You may define your own stages. They must be greater than SEOS_EXITR_MINSTAGE. gstage is undefined when authorization is not granted.

int ShouldLog

Flag indicating whether eTrust AC should record this event in the log file. A value of 0 (FALSE) indicates that logging is not required. A value of 1 (TRUE) indicates that logging is required.

int logreason

Flag indicating reason logging is required. You may define your own reasons. They must be greater than SEOS_EXITR_EXLOGMIN.

char fname[ ]

Name of source file reporting an error to the error log (__FILE__ macro in ANSI-C). This value is not used if the function returns 0.

int lnum

Line number in source file at which an error being logged in the log file occurred (__LINE__ macro in ANSI-C). This value is not used if the function returns 0.

SEOSDB_ODF

Valid for UNIX only. The SEOSDB_ODF structure contains the definition of a specific object in the database.

SEOS_CID sCId

The class ID of the object's class.

SEOS_OID lOId

The ID of the object.

char *xzOName

The name of the object.

char reserved

Reserved for future use.

SEOSDB_PDF

Valid for UNIX only. The SEOSDB_PDF structure contains the definition of a specific property in the database.

SEOS_CID sCId

The class ID of the class to which the object containing this property belongs.

SEOS_PID sPId

The ID of the property.

char *szPName

The name of the property.

unsigned long lPFlags

The flags of the property.

unsigned short sPVsize

The size in bytes of the property value.

unsigned char cPType

The type of the property value.

unsigned char cPRLevel

Not used.

unsigned char cPWLevel

Not used.

unsigned char cSegment

Not used.

char reserved

Reserved for future use.

PFSeosExitFunc

The PFSeosExitFunc data type is a pointer to a function.

void *data_buffer

Contains information about a specific event for which the exit function was called.

SEOS_EXITRES *p_sexr

Contains the results of the exit function.

int

The value returned by the authxapi function. Values for UNIX include the following:

1 AUTHXAPI_E_EINVAL

Invalid (NULL) pointers

2 AUTHXAPI_E_NOCLASS

Required class not found

3 AUTHXAPI_E_NOOBJ

Required object not found

4 AUTHXAPI_E_DBERROR

Suspect corruption of database

5 AUTHXAPI_E_INVOBJ

Invalid object descriptor

6 AUTHXAPI_E_INVPROP

Invalid property descriptor

7 AUTHXAPI_E_NOPROP

Required property not found

8 AUTHXAPI_E_PTYPE

Property type is not a list

9 AUTHXAPI_E_NOVAL

No value for property associated with this object

10 AUTHXAPI_E_NOHANDLE

Invalid eTrust AC handle

11 AUTHXAPI_E_NOACEE

No ACEE for this handle

12 AUTHXAPI_E_OCCUPIED

Exit function already installed

13 AUTHXAPI_E_NOEVENT

No such event

SEOS_CID

Valid for UNIX only.

SEOS_CID is an unsigned integer that represents the class ID. Each class in the database has a unique class ID.

SEOS_OID

Valid for UNIX only.

The SEOS_OID data type is an unsigned long integer representing the object ID of a record in the database.

Each object in the database has a unique object ID. If you know the object ID, you can use seadmapi to retrieve information about the object.

SEOS_PID

Valid for UNIX only.

SEOS_PID is an unsigned short integer representing a property ID. Each property in the database has a unique property ID.

Chapter 4: LogRoute API

This section contains the following topics:

Programming Guide

The LogRoute API lets you add your own alerts to the standard audit log functions. You can also use the log routing daemon to add a guaranteed-delivery feature to your other programs. This chapter provides details of the configuration file read by selogrd, the structures and functions used when writing a new LogRoute API, the compile and link procedures used under most operating systems, and a sample LogRoute API function.

The LogRoute API lets you insert your own alerts in the audit log file. The seosd daemon generates audit information and stores it in the audit log file. The log routing daemon selogrd polls the audit log file and sends selected local audit log records to the destination targets listed in the eTrust AC configuration file. Destination targets may be screen or mail messages to an individual user, a local system file, or files located on remote host systems on the network.

The LogRoute API lets you customize the log routing daemon selogrd. You can incorporate your own user-defined options into selogrd to support in-house requirements not provided by the standard log routing functions. You can add new target types to the configuration file read by selogrd. Add your new LogRoute API functions to the eTrust AC system by creating your own shared library that uses the eTrust AC API. You can also use the log routing daemon to add a guaranteed-delivery feature to your other programs.

The LogRoute API saves you a tremendous amount of work by letting you take advantage of all the services that the log routing daemons already provide. Regardless of the target type, whether built-in or user-defined, the log routing daemons automatically read the audit files, filter the entries to capture the records requested by the user, and store or distribute that selected information appropriately. You can also designate different targets for the configuration file and use selogrd to provide a guaranteed-delivery service to those targets for your own programs.

eTrust AC also provides an API for the daemon that collects the data from multiple stations and maintains the central audit log file.

Customizing selogrd

You add user-defined features to the log routing daemons by writing C-language programs that can be compiled and linked into a shared library. A LogRoute API function has three parts:

- Registration
- Implementation
- Termination

Registration initializes your LogRoute API function and registers it with the eTrust AC daemons. Implementation adds your tasks to the standard log routing daemon process. Termination unregisters and shuts your program down properly when the daemons themselves terminate.

Your LogRoute API function takes advantage of functions and header files provided by eTrust AC. See LogRoute API Functions in this chapter for a description of the predefined functions used in a log routing function. You use the same registration, implementation, and termination functions for all your log routing functions.

Once your LogRoute API function is ready, you can add your shared library to the log routing daemon configuration file. For more information about compiling and linking procedures, see Compiling and Linking with the LogRoute Library in this chapter.

The log routing daemons use a configuration file to determine which audit log records to select and where to send those records. You can edit the configuration file to route specific audit information to a variety of selected targets supported by the log routing daemon. For more information about the syntax of the configuration file, see the selogrd utility in the UNIX *Utilities Guide*.

LogRoute API Functions

Your LogRoute API function uses built-in functions and header files provided by eTrust AC, which provides the following predefined selogrd functions:

- driver_RegisterDestination
- driver_UnregisterDestination
- lograpi_InterpretRecord
- lograpi_MakeStringMessage
- lograpi_RegisterTargetType
- lograpi_UnregisterTargetType

eTrust AC provides the following predefined selogrcd functions:

- driver_Register
- driver_UnRegister
- servlog_IsThereExit
- servlog_RegisterExit
- servlog_UnRegisterExit

All LogRoute API functions must also include the following destination functions for each destination type implemented:

- LogrApiSenseFunc
- LogrApiSendFunc
- LogrApiFreeFunc

These three functions, which are grouped together in the LOGRAPI_FUNCS structure, are accessed using the pfSend, pfFree, and pfSense pointers. The API programmer must provide the code used for each of these functions, because each one is completely task-dependent.

Compiling and Linking with the LogRoute Library

This section provides instructions for compiling and linking your LogRoute API function with the eTrust AC daemons. These are general instructions that describe the most common system configurations. Each system has its own specific requirements. Consult your system guides for the exact details of your particular system's compiler and linker.

Compiling an Application

You must include the header files *lograpi.h* and *selogtype.h* in your LogRoute API functions. These files are located in the API subdirectory. Put the following two lines near the top of the file:

```
#include <lograpi.h>
#include <selogtype.h>
```

You can use any ANSI-C compliant compiler.

Linking Applications with the LogRoute Library

After you compile your code, generate a shared library that contains the compiled version of your function. The apisamples directory contains sample log routing functions and a makefile demonstrating the process. Note that compilation for shared libraries usually requires additional compiler parameters to create position-independent code. See the documentation for your compiler or linker for information on creating shared libraries in your particular system.

After you have written your code and created a shared library, add your shared library to the "on-demand" shared libraries configuration file relevant to the daemon your code should link to.

If you have written a shared library for selogrd, add your shared library to the file *eTrustACDir*/etc/selogrd.ext. If you have written a shared library for selogrcd, add your shared library to the file *eTrustACDir*/etc/selogrcd.ext.

Each file contains two columns: the driver name and the shared library path. By convention, the driver name is a string that has the same name as your target type; however, it can be any valid C language symbol.

For example, if you have written code to implement a pager, your target name should be *pager* and the complete file entry would be:

```
pager /usr/local/lib/libseospager.so
```

This file entry means that the daemon selogrd loads the shared library /usr/local/lib/libseospager.so at startup and calls your function.

Although some systems support a predefined function called _init, we recommend that you use the function driver_RegisterDestination. This is the first function called from the shared library.

The function driver_RegisterDestination registers your new target type. On daemon shutdown, we recommend that you use the function driver_UnregisterDestination instead of the predefined _fini.

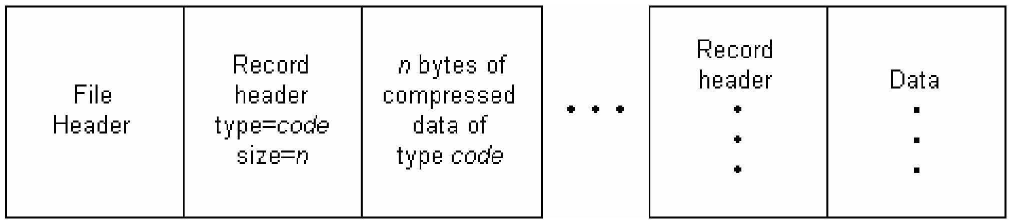
Note: Using the eTrust AC functions instead of the predefined system functions gives your code greater portability

The daemon selogrcd uses the same file configuration format as selogrd. However, selogrcd searches for the driver_Register and driver_UnRegister functions. If the function driver_UnRegister is not required, it can be omitted.

Format of the Log File

The format in which eTrust AC writes the auditing log is not publicly available. However, it is essential that programmers who use this API understand the basics of the audit log file format. The information programmers must know is provided in this section.

The audit log file is composed of a file header followed by records sequentially written to the file. Each record is composed of a record header followed by information specific to the record. The header of each record includes information about the time that the record was placed in the file, the record type (a code known only by the application), and the size of the record that follows the header. The data itself is written in a compressed format; the record size specified in the record header is the size in bytes of the compressed data. The format of the file is shown schematically in the following diagram.


File Header
Record header type=*code* size=*n*
n bytes of compressed data of type *code*
• • •
Record header
Data

The programmer using this API does not need to know the compression algorithm or the exact format of the file. Information passed to a user application is placed in structures in uncompressed format. Your application simply retrieves the information from the structure. For more information about the LogRoute API structures, see Structures and Data Types in this chapter.

LogRoute API Example

The API subdirectory under the local eTrust AC directory contains the API header files and the library functions. The eTrust AC package also includes the following sample program demonstrating LogRoute API use-adding the destination target syslog to eTrust AC to send audit information to UNIX system logs.

```
/*==========================================================
Project      : eTrust
Module       : eTrust
Version      : 8.0
File         : audit2syslog.c
Purpose      : Provide sample usage of the selogrd API to place
               audit log records in UNIX syslog.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
============================================================*/
#define __LOGRSAMPLE_C
#include <syslog.h>
#include <lograpi.h>

/* Include the .h file required by API */
/* Prototypes for our local functions */
static int sample_Sense(SEOS_ROUTENTRY *pre);
static void sample_Free(SEOS_ROUTENTRY *pre);
static int sample_Send(LOGRECORD *plr, SEOS_ROUTENTRY *pre,
                       int notify, void *data);
/*
   * We don't use the code of the new route type target, but
   * if we wanted more than one target type, we could use
   * it to distinguish between the two.
*/
static int our_dest_type_code;
/*
   * We preserve here the syslog priority required by
   * the configuration file.
   * This of course means that by storing it in global
   * variable we provide only one route line in the
   * configuration file for syslog. Other lines will just
   * overwrite this variable.
*/
static int syslog_priority;
/*
   * This like our 'main'. This function is the one called by
   * the selogrd.
*/
int lograpi_RegisterDestinations(void)
{
    static LOGRAPI_FUNCS funs =
    {sample_Send, sample_Free, sample_Sense};
    /*...*/
    return lograpi_RegisterTargetType("syslog", &funs,
                                      &our_dest_type_code);
}
static int sample_Sense(SEOS_ROUTENTRY *pre)
```

```
{
    /* Actually we have nothing to do, since syslog
     * should be there.
     * Anyway, just to demonstrate what to do we will look
     * at the destination name in this type of function,
     * we will check that the destination is one of the
     * known syslog priorities.
     */
    typedef struct tagAllowedDestNames

    {
    char const *name;
    int code;
    } ALLOWED_DEST_NAMES;
    static ALLOWED_DEST_NAMES allowed_names[] =
    {
    {“LOG_EMERG”, LOG_EMERG},
    {“LOG_ALERT”, LOG_ALERT},
    {“LOG_CRIT”, LOG_CRIT},
    {“LOG_ERR”, LOG_ERR},
    {“LOG_WARNING”, LOG_WARNING},
    {“LOG_NOTICE”, LOG_NOTICE},
    {“LOG_INFO”, LOG_INFO},
    {“LOG_DEBUG”, LOG_DEBUG},
    {NULL, 0}
    };
    register int i;
    for (i = 0; allowed_names[i].name != NULL; i++) {
                if (strcmp(allowed_names[i].name, pre->out) == 0) {
                    /* Preserve the method we should use in syslog */
                        syslog_priority = allowed_names[i].code;
                        return 0;
                    }
    }
    return 1;
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
static void sample_Free(SEOS_ROUTENTRY * pre)
{
    /*...*/      /* Now really nothing to be done */
}

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
static int sample_Send(LOGRECORD * plr, SEOS_ROUTENTRY * pre,
                         int notify, void *data)
{
    char *as_string;

    if (notify)           /* Ignore any NOTIFY messages */
```

```
                return 0;
    as_string = lograpi_MakeStringMessage(plr, data);
    if (as_string != NULL)
                syslog(syslog_priority, as_string);
    return 0;
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
```

Notification Audit Log Records

eTrust AC lets you store notification information in the database as a string associated with a user or resource record. The administrator can specify mail addresses to be notified each time an attempt is made to access the resource. A notification request is stored as a special audit log record in the audit log file. selogrd routes the notification request to the mail or the screen address of the destination specified in the audit log record.

Notification records for a given event are identical to the standard audit log records associated with that event, except that notification records also have their targets stored at the beginning of the audit log record. The log codes for the notification records are simply the log codes of regular audit log records, offset by 2048. For example, a normal login audit log record has a log type code of 1. The notification log type code would be 2049. Note that an audit log record can appear in the audit log file followed by the notification record of the same event.

The format of the structure name a notification record is SEOSNF_AUDIT*. The exact names correspond to the matching audit log record name:

Notification Record	Audit Log Record
SEOSNF_AUDITADMIN	SEOS_AUDITADMIN
SEOSNF_AUDITGENR	SEOS_AUDITGENR
SEOSNF_AUDITINWARN	SEOS_AUDITINWARN
SEOSNF_AUDITLOGIN	SEOS_AUDITLOGIN
SEOSNF_AUDITWDWARN	SEOS_AUDITWDWARN

In each structure, the first field is SEOS_NOTIFYSTR, a buffer of up to 30 bytes to hold the destination string pulled from the database. The second field is the audit log record corresponding to this notification record.

LogRoute API Function

eTrust AC provides the following functions:

driver_Register

Must be in your shared library and is called when selogrcd starts.

driver_UnRegister

Must be in your shared library and is called when selogrcd shuts down.

driver_RegisterDestination

Must be in your shared library and is called when selogrd starts.

driver_UnregisterDestination

Must be in your shared library and is called when selogrd shuts down.

lograpi_Interpret_Record

Converts an audit log record to the vector of text string pairs.

lograpi_MakeStringMessage

Converts an audit log record structure to the one-line text format used by seaudit and seauditx.

lograpi_RegisterTargetType

Informs eTrust AC of the exact implementation details of the new target types being registered.

lograpi_UnregisterTargetType

Removes a destination type that was previously registered with the log routing daemon.

servlog_IsThereExit

Tests whether an exit function exists.

servlog_RegisterExit

Registers an exit function.

servlog_UnRegisterExit

Unregisters an exit function.

Your LogRoute API functions that implement a new destination type must supply code for the following tasks:

LogrApiFreeFunc

Frees any memory allocated by the user code for storage, sockets, and so on, and closes all network connections.

LogrApiSendFunc

Sends the selected audit log record to the user-specified target.

LogrApiSenseFunc

Tests the target addresses in each configuration file line for correctness.

driver_Register Function

The *driver*_Register function is a predefined function called by the selogrcd daemon when it starts. You insert your own code into *driver*_Register to register all your new customized destination types. The function should register the exit functions needed for each audit record type.

Replace *driver* with the destination type as it appears in both the route configuration file and extension configuration file of selogrcd.

```
int driver_Register (void);
```

If the function succeeds, it should return 0. If it fails, it returns a nonzero integer error code. This return code can be seen in syslog. If selogrcd is using debug mode, the return code can also be seen on the screen.

More information:

driver_UnRegister Function (see page 119)
lograpi_RegisterTargetType Function (see page 123)

driver_UnRegister Function

The *driver*_UnRegister function is a predefined function called by the selogrcd daemon when it shuts down. You insert your own code into *driver*_UnRegister to unregister all the destination types you registered.

Replace *driver* with the destination type as it appears in both the route configuration file and extension configuration file of selogrcd.

```
int driver_UnRegister (void);
```

If the function succeeds, it should return 0. If it fails, it returns a nonzero integer error code. This return code can be seen in syslog. If selogrcd is using debug mode, the return code can also be seen on the screen.

More information:

driver_Register Function (see page 119)

driver_RegisterDestination Function

The *driver*_RegisterDestination function is a predefined function called by the selogrd daemon when the daemon starts. You insert your own code into *driver*_RegisterDestination to register all your new customized destination types.

Replace *driver* with the destination type as it appears in both the route configuration file and extension configuration file of selogrd.

The function is called when the shared library is loaded. The function should initialize the library and register the new target type by calling the function lograpi_RegisterTargetType.

```
int driver_RegisterDestination (void);
```

If the function succeeds, it should return 0. If it fails, it returns a nonzero integer error code. This return code can be seen in syslog. If selogrd is using debug mode, the return code can also be seen on the screen.

More information:

driver_UnregisterDestination Function (see page 121)
lograpi_RegisterTargetType Function (see page 123)

driver_UnregisterDestination Function

The *driver*_UnregisterDestination function is a predefined function called by the selogrd daemon at system termination. You insert your own code into *driver*_UnregisterDestination to unregister all the destination types you registered.

Important! Once you unregister a destination type, it cannot be registered again during the current session. However, all subsequent records are marked as if the send to that destination was successful. Therefore, do not unregister a target type unless you definitely will not be working with that target type again during the current session.

Replace driver with the destination type as it appears in both the route configuration file and extension configuration file of selogrd.

The function is called when the shared library is loaded. The function should initialize the library and register the new target type by calling the function lograpi_RegisterTargetType.

```
int driver_UnregisterDestination (void);
```

If the function succeeds, it should return 0. If it fails, it returns a nonzero integer error code. This return code can be seen in syslog. If selogrd is using debug mode, the return code can also be seen on the screen.

More information:

driver_RegisterDestination Function (see page 120)

lograpi_InterpretRecord Function

The lograpi_InterpretRecord function converts an audit log record to a vector of text string pairs. Each pair consists of the label of the field in the record and the text for that field. The vector itself ends when both members-label and value-are NULL pointers. The value can be a NULL pointer when a specific field has no value in the audit record.

If the function succeeds, it returns a pointer to the vector of structures. Each element in the vector is a structure that contains two members: a label and a value. The pointers Label and Value point to a static memory region that is overwritten by any call to the function. The vector itself ends when both members are NULL pointers.

The *seauditx* utility displays audit records in a format similar to this when the user requests more detail about displayed records.

```
SEOS_AUDLOGINTERP * lograpi_InterpretRecord P(LOGRECORD *plr, void *unc_buff);
```

plr

A pointer to the audit log record structure passed to the LogrApiSendFunc function.

unc_buff

A pointer to the uncompressed audit log record information passed to the LogApiSendFunc function.

More information:

lograpi_MakeStringMessage Function (see page 125)

lograpi_RegisterTargetType Function

The lograpi_RegisterTargetType function registers a new target or destination type with the log routing daemon. lograpi_RegisterTargetType provides the LOGRAPI_FUNCS structure with pointers to the three user functions used to sense a valid configuration file entry, send the record, and free the allocated memory space. lograpi_RegisterTargetType is normally called by the driver_RegisterDestination function, to register your functions with the log router.

If the function succeeds, it returns 0. If it fails, it returns an unsigned integer error code and assigns a value to the global variable errno according to the following table of values:

Return Value	ERRNO	Meaning
LOGRAPI_E_DESTFULL	ENOMEM	Destination table is full; maximum table size is 10 elements.
LOGRAPI_E_NULLPARAM	EINVAL	One of the parameters is NULL.
LOGRAPI_E_NOSENDFUNC	EINVAL	No send function specified.

```
int lograpi_RegisterTargetType (const char *name, LOGRAPI_FUNCS *funcs, int
*code);
```

name

The name of the newly added destination type.

funcs

A pointer to a LOGRAPI_FUNCS structure containing the three destination type functions, LogrApiSendFunc, LogrApiFreeFunc, and LogrApiSenseFunc.

code

The code assigned to this target destination. The code is the value stored in the destination data member of the SEOS_ROUTENTRY structure.

More information:

driver_RegisterDestination Function (see page 120)
lograpi_UnregisterTargetType Function (see page 124)

lograpi_UnregisterTargetType Function

The lograpi_UnregisterTargetType function unregisters a target or destination type previously registered with the log routing daemon.

If the function succeeds, it returns 0. If it fails, it returns an unsigned integer error code and assigns a value to the global variable errno according to the following table of values:

Return Value	ERRNO	Meaning
LOGRAPI_E_NULLPARM	EINVAL	The code parameter is NULL.
LOGRAPI_E_NODEST	ENOENT	No such destination type.

IMPORTANT! Once a destination type is unregistered, it cannot be registered again during the current session. However, all subsequent records are marked as if the send to that destination was successful. Do not unregister a target type unless you definitely will not be working with that target type any more.

```
int lograpi_UnregisterTargetType (const char *name);
```

name

The name of the target or destination type to be unregistered.

More information:

driver_UnregisterDestination Function (see page 121)
lograpi_RegisterTargetType Function (see page 123)

lograpi_MakeStringMessage Function

The lograpi_MakeStringMessage function converts an audit log record to a one-line text string in the standard format used by the seaudit and seauditx utilities.

If the function succeeds, it returns a char pointer to the audit log data string. This string is held in an area of static memory that is overwritten when a subsequent call is made to the function.

If the function fails, it returns NULL. Check the return value of errno for more information. Passing a NULL pointer as an input parameter generates an error. Possible errors are:

Return Value	Meaning
EINVA	Data on pointer parameters are NULL.

```
char * lograpi_MakeStringMessage (LOGRECORD *plr, void *data);
```

plr

A pointer to the audit log record structure passed to the LogrApiSendFunc function.

data

A pointer to the uncompressed audit log record information passed to the LogrApiSendFunc function.

More information:

lograpi_InterpretRecord Function (see page 122)

LogrApiSenseFunc Function

LogrApiSenseFunc is a function pointer type that specifies a user-defined sense function to be called while the selogrd daemon is initializing and restarting. The sense function determines (senses) if the configuration file route entry is valid. The sense function tests the target field entries in each configuration file line for validity. For example, if the destination name is user jsmith, then LogrApiSenseFunc should check that there is, in fact, a user by that name.

If the function succeeds, it should return 0. If it fails, it returns an error code.

```
typedef int (*LogrApiSenseFunc) (SEOS_ROUTENTRY *pre);
```

pre

Configuration file entry to check for validity.

LogrApiSendFunc Function

LogrApiSendFunc is a function pointer type that specifies a user-defined send function. When an audit log record is found that matches the user's selection criteria, the send function transmits (sends) the selected audit log record to the user-specified destination.

If the send action succeeds, the function should return 0. An audit log record successfully sent is never submitted again to LogrApiSendFunc.

If the send action fails, eTrust AC enters an error notice into the syslog file and returns a nonzero integer as an error flag. The same audit log record may be resubmitted an unlimited number of times.

```
typedef int (*LogrApiSendFunc) (LOGRECORD *plr, SEOS_ROUTENTRY  *pre, \
                                            int  notify,  void *data);
```

plr

The audit log description file.

pre

Information about the audit target destination for the audit log record.

notify

Flag indicating if this audit log record is a notification record as follows:

1 (TRUE)

Audit log record is a notification record.

0 (FALSE)

Audit log record is not a notification record.

data

A pointer to the audit log record.

More information:

LogrApiFreeFunc Function (see page 127)

LogrApiFreeFunc Function

LogrApiFreeFunc is a function pointer type that specifies a user-defined free function. When selogrd shuts down or restarts, it calls the free function to free the memory allocated to a previously registered function. The SendData member of the SEOS_ROUTENTRY structure is used as a placeholder for the allocated memory for a target, such as a FILE * or a CLIENT *. The pointer may be NULL, if the registered function uses no allocated resources.

The selogrd daemon shuts down and restarts every time seosd switches log files. This happens often; for example, whenever the log files exceed a specified maximum size. Be sure that your free function reliably frees all allocated memory or you may create problems on your system.

There is no return value.

If you do not need a free operation, set this function pointer to NULL.

```
typedef void (*LogrApiFreeFunc) (SEOS_ROUTENTRY *pre);
```

pre

The target entry to free or close.

servlog_IsThereExit Function

The servlog_IsThereExit function tests if an exit function is registered for the given type of audit record. Each audit record is defined as a particular record type, such as login, audit, or general resource. The values for rectype are defined in the header file selogtype.h, which is supplied with the eTrust AC API. The format of the rectype is AUDIT_rectype.

The function returns 1 if an exit function for the specified record type exists; otherwise, it returns 0.

```
int servlog_IsThere Exit (int rectype);
```

rectype

Is an exit function registered for the record type represented by this particular code?

servlog_RegisterExit Function

The servlog_RegisterExit function registers an exit function to be called by the selogrcd daemon when a particular type of audit record is received. Each audit record in eTrust AC is identified as a particular record type, such as login, audit, or general resource. The values for rectype are defined in the header file selogtype.h, which is supplied with the eTrust AC API. The format of the rectype is AUDIT_rectype.

It is possible to register more than one exit function for each type of record; eTrust AC allows a maximum of 16 exit functions for each type of record. When a function is registered, it is assigned a sequence number in the list of exit functions for its particular type.

This function should be called during exit initialization to register the exit functions. This function is normally called from the driver_Register function.

The function returns a 0 on success and an error code on failure.

```
int servlog_RegisterExit (int rectype, collectexitf func, int *chain);
```

rectype

The code of the record type for which the exit function must be called.

func

A pointer to the user function that should gain control when an audit record of rectype is received.

chain

The number in the chain of exit functions of the specified record type.

servlog_UnRegisterExit Function

The servlog_UnRegisterExit function unregisters an exit function previously registered by a call to servlog_RegisterExit. After unregistering an exit function, it can no longer be called.

The function returns a 0 on success and an error code on failure.

```
int servlog_UnRegisterExit (int rectype, int *chain);
```

rectype

The code of the audit record type.

chain

The number (from the chain of exit functions for the specified record type) of the exit function to be unregistered. The number was assigned to the function when it was registered.

Structures and Data Types

This section describes the data structures used by LogRoute API functions to pass information back and forth between the functions and the eTrust AC daemons (in UNIX) and services (in Windows). Every field of each data structure is described.

The LogRoute API functions access two types of data structures: audit log record structures and information-passing structures.

Audit Log Record Structures

There are eight types of audit log records, each with its own structure format. The structures can be found in the source file selogtype.h. The following audit log record structures are included in the LogRoute API:

SEOS_AUDITADMIN

Used for database update events

SEOS_AUDITDOWN

Used for daemon shutdown events

SEOS_AUDITGENR

Used for general resource check events

SEOS_AUDITINWARN

Used for TCP/IP request events

SEOS_AUDITLOGIN

Used for login events

SEOS_AUDITSTART

Used for daemon startup events

SEOS_AUDITUSER

Used for user trace events

SEOS_AUDITWDWARN

Used for Watchdog events

The LogRoute API includes data structures used to pass audit log records, configuration file information, and LogRoute API function pointers between the eTrust AC daemons and your LogRoute API functions. Details of the four data structures used to pass information between functions are given in the second half of this section. The structures themselves can be found in the source file lograpi.h. The four information-passing structures are as follows:

LOGRECHDR

Stores the audit log record header

LOGRECORD

Stores the audit log record data

SEOS_ROUTENTRY

Stores the configuration file entry

LOGRAPI_FUNCS

Contains LogRoute API implementation functions

Notification Audit Log Records

eTrust AC lets you store notification information in the database as a string associated with a user or resource record. The administrator can specify mail addresses to be notified each time that an attempt is made to access the resource. A notification request is stored as a special audit log record in the audit log file. selogrd routes the notification request to the mail or the screen address of the destination specified in the audit log record.

Notification records for a given event are identical to the standard audit log records associated with that event, except that notification records also have their targets stored at the beginning of the audit log record. The log codes for the notification records are simply the log codes of regular audit log records, offset by 2048. For example, a normal login audit log record has a log type code of 1. The notification log type code would be 2049. Note that an audit log record can appear in the audit log file followed by the notification record of the same event.

Notification record structure names begin with SEOSNF_AUDIT. The exact names correspond to the matching audit log record name:

SEOSNF_AUDITADMIN

SEOS_AUDITADMIN

SEOSNF_AUDITDOWN

SEOS_AUDITDOWN

SEOSNF_AUDITGENR

SEOS_AUDITGENR

SEOSNF_AUDITINWARN

SEOS_AUDITINWARN

SEOSNF_AUDITLOGIN

SEOS_AUDITLOGIN

SEOSNF_AUDITSTART

SEOS_AUDITSTART

SEOSNF_AUDITUSER

SEOS_AUDITUSER

SEOSNF_AUDITWDWARN

SEOS_AUDITWDWARN

In each structure, the first field is SEOS_NOTIFYSTR, a buffer of up to 30 bytes to hold the destination string pulled from the database. The second field is the audit log record corresponding to this notification record.

SEOS_AUDITLOGIN

The SEOS_AUDITLOGIN record may be submitted to the audit log file when:

- A user logs in, attempts to log in, or logs out
- serevu disables or enables a user
- A user fails to log in after a certain number of attempts
- seosd detects an attack on the network

If the access and use of a resource are being monitored, audit records are also submitted to the audit log. Logout audit records are submitted to the log file only if a login record was also submitted.

char szUserName[ ]

Name of user logging in (ASCII-Z string).

char szTerminal[ ]

Name of terminal or network host from which user is logging in (ASCII-Z string).

int LogCode

Reason this audit log record was added to the file. There are several possible reasons for eTrust AC to record a login attempt. See Login Event Codes in this chapter for more information.

int stage

Stage in the authorization algorithm when the decision was made to grant or deny access. The LogRoute API includes a listing of these stage codes in the header file seauthstages.h.

uid_t uid

User's UNIX or Windows user ID.

char szProg[ ]

Name of the program attempting to perform the login.

Login Event Codes

The SEOS_AUDITLOGIN record is used to audit login attempts, logouts, auditing and serevu login attempts, and NAP detection by seosd. Login audit records are submitted to the log for the following reasons:

SEOS_AUTH_PASS

The user was allowed to log in.

SEOS_AUTH_DENY

The user was denied login access.

SEOS_AUTH_CHECK

An error in the database was found.

Logout audit records are submitted to the log file only if a login record was also submitted. That is, the user's audit mode includes the auditing of successful logins, or the terminal from which the user logged in has an audit mode that includes the auditing of successful accesses. Logout records are assigned the stage code SEOS_LOGOUT_RES.

When serevu detects attempts to log in, the following reason codes apply:

SEOS_LOGATP_RES

Detected attempt to break password.

SEOS_LOGDIS_RES

The specified user account was disabled by serevu because of too many login attempts.

SEOS_LOGENA_RES

The specified user account was reactivated by serevu after being disabled for the configured time period.

In all cases, the stage code assigned to login records written by serevu is SEOS_LOG_SEREVU.

SEOS_AUDITGENR

The SEOS_AUDITGENR record can be submitted to the audit log file when a user accesses, or attempts to access, a general resource.

char szUserName[ ]

Name of user attempting to gain access (ASCII-Z string).

char szResClass[ ]

Class of resource being accessed (ASCII-Z string).

char szResource[ ]

Name of resource being accessed (ASCII-Z string).

int logReason

Reason this audit log record was added to the file. Either the user or the resource involved has been flagged for auditing. Possible reasons are listed in the table in Return Codes in this chapter. In UNIX, the LogRoute API also includes a listing of these reasons in the header file seauthstages.h. In Windows, you can find this information in the following directory of your system drive:

eTrustACDir\include

int stage

Stage in the authorization algorithm when the decision was made to grant or deny access. The LogRoute API includes a listing of these stage codes in the header file seauthstages.h.

SEOS_ACCS access

User's level of access to the resource. For a list of available access types, see the file seostype.h or the SEOS_ACCS data type in the chapter "Authorization and Authentication API."

uid_t uid

User's UNIX or Windows user ID.

char szProg[ ]

Name of the program that attempted to gain access to the resource (ASCII-Z string).

char szTerm[ ]

Name of terminal or network host from which user logged in (ASCII-Z string).

Reason Code	Integer Value	Meaning
WDWARN_ERROR	0	An error occurred

Reason Code	Integer Value	Meaning
WDWARN_STATCHANGED	1	Stat was changed
WDWARN_AIXEXIT	2	HP-UX/AIX extended information changed
WDWARN_AIXACL	3	HP-UX/AIX ACL changed
WDWARN_CRC	4	CRC check failed
WDWARN_STAT	5	Cannot obtain information about the trusted file
WDWARN_MD5	6	MD5 signatures do not match

SEOS_AUDITWDWARN

The SEOS_AUDITWDWARN record can be submitted to the audit log file when the Watchdog (seoswd) finds an integrity problem in a trusted program or a secured file.

char szClass[ ]

Class name of resource being audited (ASCII-Z string). This can be PROGRAM or SECFILE.

char szPath[ ]

Full path name of the program or secure file being audited (ASCII-Z string).

int errno

System errno value that may have triggered this audit.

int logReason

Reason this audit log record was added to the file. Possible reasons are listed in the table in Return Codes in this chapter. In UNIX, the LogRoute API also includes a listing of these reasons in the header file seauthstages.h. In Windows, you can find this information in the following directory of your system drive:

eTrustACDir\include

int stage

Stage in the authorization algorithm when the decision was made to grant or deny access. The LogRoute API includes a listing of these stage codes in the header file seauthstages.h.

Reason Code	Integer Value	Meaning
WDWARN_ERROR	0	An error occurred
WDWARN_STATCHANGED	1	Stat was changed
WDWARN_AIXEXIT	2	HPUX/AIX extended information changed
WDWARN_AIXACL	3	HPUX/AIX ACL changed
WDWARN_CRC	4	CRC check failed
WDWARN_STAT	5	Cannot obtain information about the trusted file
WDWARN_MD5	6	MD5 signatures do not match

SEOS_AUDITINWARN

The SEOS_AUDITINWARN record can be submitted to the audit log file when a remote host attempts access to the local host and that remote host has been flagged for auditing.

char address[20]

Internet address of the remote host attempting access. This is currently the 4-byte address of TCP.

char af_type

AF number. Currently, only AF_INET (2).

long port

Port number to which access was attempted.

long proto

Protocol code. Currently 0.

char szProg[ ]

Name of the program in the local host that was trying to accept the access request.

int logReason

Reason this audit log record was added to the file. Possible reasons are listed in the table in Return Codes in this chapter. In UNIX, the LogRoute API also includes a listing of these reasons in the header file seauthstages.h. In Windows, you can find this information in the following directory of your system drive:

eTrustACDir\include

int stage

Stage in the authorization algorithm when the decision was made to grant or deny access. The LogRoute API includes a listing of these stage codes in the header file seauthstages.h.

Reason Code	Integer Value	Meaning
WDWARN_ERROR	0	An error occurred
WDWARN_STATCHANGED	1	Stat was changed
WDWARN_AIXEXIT	2	HPUX/AIX extended information changed
WDWARN_AIXACL	3	HPUX/AIX ACL changed
WDWARN_CRC	4	CRC check failed

Reason Code	Integer Value	Meaning
WDWARN_STAT	5	Cannot obtain information about the trusted file
WDWARN_MD5	6	MD5 signatures do not match

SEOS_AUDITADMIN

The SEOS_AUDITADMIN record can be submitted to the audit log file when a selang command updates the database.

char szClass [AUDITADMIN_MAXCLASSLEN]

Class on which the operation was performed.

char objname [ONAME_SIZE+1]

Object on which the operation was performed.

char user [ONAME_SIZE+1]

Administrator who issued the command.

int reason

Reason this audit log record was added to the file. Possible reasons are listed in the table in Return Codes in this chapter. The LogRoute API also includes a listing of these reasons in the header file seauthstages.h.

int stage

The stage in the authorization algorithm when the decision was made to grant or deny the request. The LogRoute API includes a listing of the eTrust AC stage codes in the header file seauthstages.h.

char terminal [AUDITADMIN_MAXTERMLEN]

The terminal from which the operation was performed.

char command [AUDITADMIN_MAXCMDLEN]

The selang command used.

Reason Code	Integer Value	Meaning
WDWARN_ERROR	0	An error occurred
WDWARN_STATCHANGED	1	Stat was changed

Reason Code	Integer Value	Meaning
WDWARN_AIXEXIT	2	HPUX/AIX extended information changed
WDWARN_AIXACL	3	HPUX/AIX ACL changed
WDWARN_CRC	4	CRC check failed
WDWARN_STAT	5	Cannot obtain information about the trusted file
WDWARN_MD5	6	MD5 signatures do not match

SEOS_AUDITSTART

The SEOS_AUDITSTART record is submitted to the audit log file when an eTrust AC daemon (in UNIX) or service (in Windows) starts.

char servname

The name of the daemon or service that was started.

SEOS_AUDITDOWN

The SEOS_AUDITDOWN record is submitted to the audit log file when an eTrust AC daemon (in UNIX) or service (in Windows) is shut down.

char szUser

The name of the user who brought the daemon or service down.

char servname

The name of the daemon or service that was shut down.

int stage

The stage in the authorization algorithm when the decision was made to grant or deny the request. The LogRoute API includes a listing of the eTrust AC stage codes in the header file seauthstages.h.

SEOS_AUDITUSER

The SEOS_AUDITUSER record is submitted to the audit log file when a trace record is written for a user whose trace actions are being audited.

char szResClass

The name of the resource class that was accessed.

char szResource

The name of the specific resource object-record-that was accessed.

int code

The trace message code.

int stage

The stage in the authorization algorithm when the decision was made to grant or deny the request. The LogRoute API includes a listing of the eTrust AC stage codes in the header file seauthstages.h.

uid_t uid

User's UNIX or Windows user ID.

uid_t euid

User's effective UID.

uid_t ruid

The ID the user used to log in-the real user ID.

char parm_buff

The parameters in the trace message.

LOGRECHDR

There are many different types of audit log records, each with its own structure format. eTrust AC has to know what type of record structure to expect for the next record; therefore, each record stored in the audit log file has a header structure common to all audit log records.

LOGRECHDR is the header structure common to all audit log records.

unsigned long nBytes

The size, in bytes, of the record in the compressed log file, not including the header.

time_t tLog

The time the record was placed in the file.

unsigned long positor

A code for the module that wrote the record. Normally, it has a value of zero.

unsigned long rectype

The record type. Valid record type codes are:

- AUDIT_LOGIN
- AUDIT_GENR
- AUDIT_WDWARN
- AUDIT_INWARN
- AUDIT_ADMIN
- AUDIT_DOWN
- AUDIT_START
- AUDIT_USER
- AUDIT_CWS

These codes are described in the selogtype.h file.

unsigned long rv

Return code that caused the record to be written to log. Possible reasons are listed in the table in Return Codes in this chapter.

Code	**Value**	**Audit Record Types**	**Description**
SEOS_AUTH_CHECK	C	All	An error occurred in eTrust AC.

Code	Value	Audit Record Types	Description
SEOS_AUTH_DENY	D	Login General Resource Admin Inet	eTrust AC denied access to a resource, did not permit a login, or did not permit an update to the database because the accessor did not have sufficient authorization.
SEOS_AUTH_PASS	P	Login General Resource Inet	eTrust AC permitted access to a resource or permitted a login.
SEOS_DOWN_RES	M	Down Start	The eTrust AC daemons started up or shut down.
SEOS_LANG_DENY	D	Admin	An attempt to update the database was denied.
SEOS_LANG_FAIL	F	Admin	An attempt to update the database failed.
SEOS_LANG_SUCC	S	Admin	The database was successfully updated.
SEOS_LOGATP_RES	A	Login	An attempt to log in failed because an invalid password was entered more than once.
SEOS_LOGDIS_RES	I	Login	The serevu daemon disabled a user.
SEOS_LOGENA_RES	E	Login	The serevu daemon enabled a disabled user.
SEOS_LOGOUT_RES	O	Login	A user logged out.
SEOS_USER_RES	T	User	An audit record written because all actions of the user are being traced.
SEOS_WATCHDOG_RES	W	Watchdog	The seoswd or seosd daemon set a program in the PROGRAM class or a file in the SECFILE class as untrusted.

LOGRECORD

The LOGRECORD structure contains the complete audit log record. The generic void *data points to any of the data structures used to hold the record data.

LOGRECHDR lrh

Log record header.

void *data

The compressed data record. Note that the user function receives this data after it is uncompressed.

SEOS_ROUTENTRY

The SEOS_ROUTENTRY structure contains the filtering and target information from each rule in the configuration file. This information is parsed by selogrd. Note that in this structure, all elements in lowercase are *read-only*, while elements in mixed case are *read-write*.

char szClass

The class name.

char obj

The object or resource name.

char accr

The accessor user name.

char code

The access result code:

(Pass)	Success
(Deny)	Failure
(Untrust)	Untrusted action was attempted on a trusted program checked by the Watchdog.

Additional values are documented in the file selogtype.h.

int dest

The destination type code. The codes are dynamically allocated as the destination types are registered.

char out

The target routing path.

void *SendData

A placeholder for information to be stored by the routing functions, such as open file handles.

int in_error

Boolean flag set if this route entry has previously failed to transmit information. selogrd calls the destination send function repeatedly to resend the audit records that failed to be transmitted.

LOGRAPI_FUNCS

The LOGRAPI_FUNCS structure contains pointers to the user-defined functions for each of the tasks to be performed by a destination type. This structure is used only during target type registration.

LogrApiSendFunc pfSend

A pointer to the user's send function.

LogrApiFreeFunc pfFree

A pointer to the user's free function.

LogrApiSenseFunc pfSense

A pointer to the user's sense function.

Chapter 5: Language Client API

This section contains the following topics:

Programming Guide

The Language Client API (LCA) lets you add your own functions on top of the eTrust AC authorization and authentication functions. It also lets you add a special notification function to the seosd daemon.

The LCA provides a high-level programming interface that you can use to administer local and remote eTrust AC databases, as well as Policy Model Databases (PMDBs). The LCA includes functions that can read and modify the values of properties stored in all these databases. It can also control the behavior of the seosd and seagent daemons.

The LCA is based on, and uses the functions of, the eTrust AC Administration API (see the chapter “Administration API”). The LCA functions use the Administration API functions to access and update the various eTrust AC databases. However, the two APIs have significant differences:

- The LCA can administer several local, remote, and PMDBs simultaneously; the Administration API can administer the database only at the local station.
- The LCA can manage features of the native operating system security (for example, passwords, group memberships, and file permissions); the Administration API manages only the eTrust AC database.

To supply detailed error information (for example, which keyword or subcommand failed), the LCA include lists of error code structures.

Note: You can execute LCA commands from the TCL shell. This lets you obtain more information about eTrust AC objects. For more information, see the appendix “tcllca: The LCA Extension.”

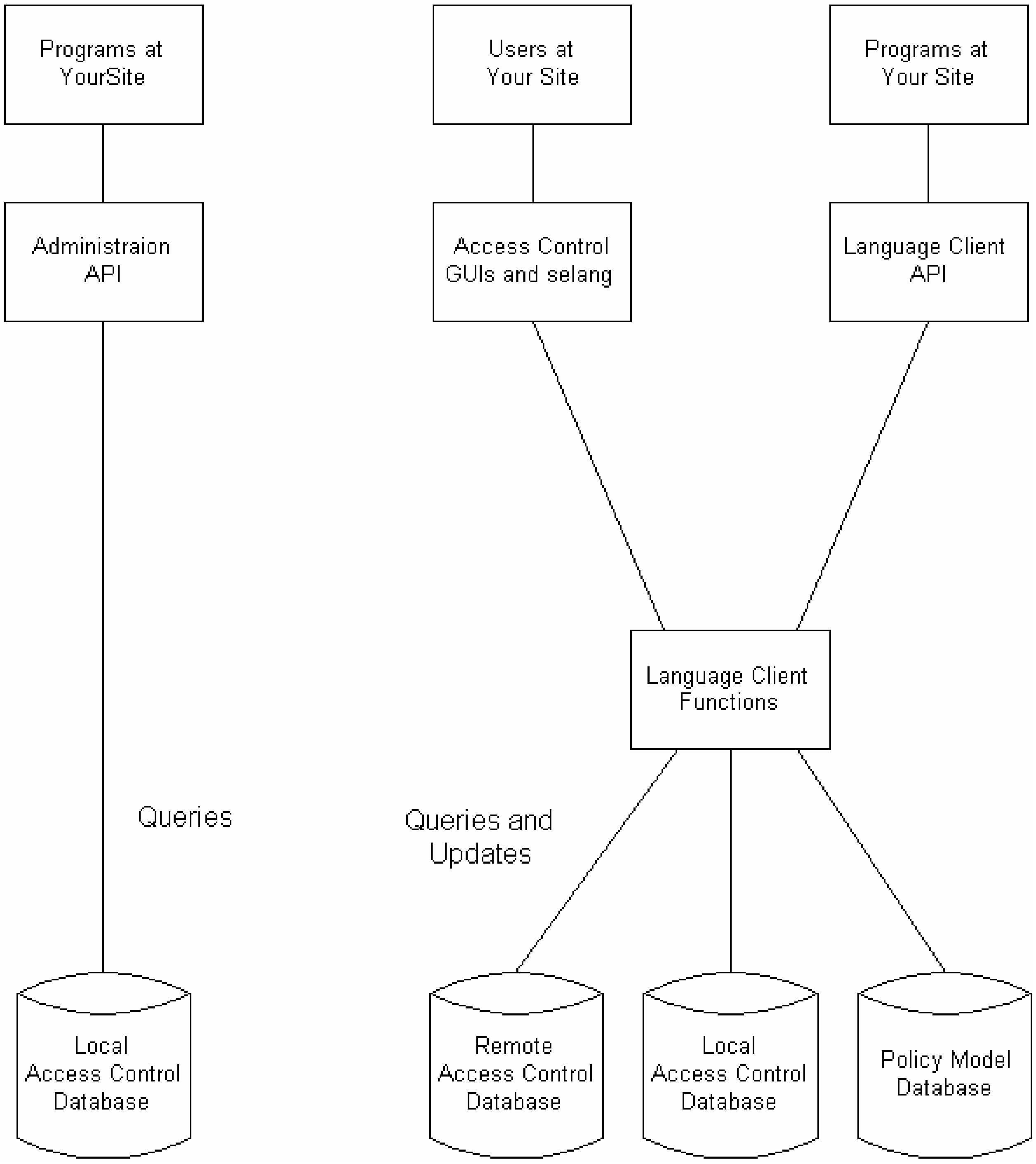
LCA Function Types

The LCA includes the following function types:

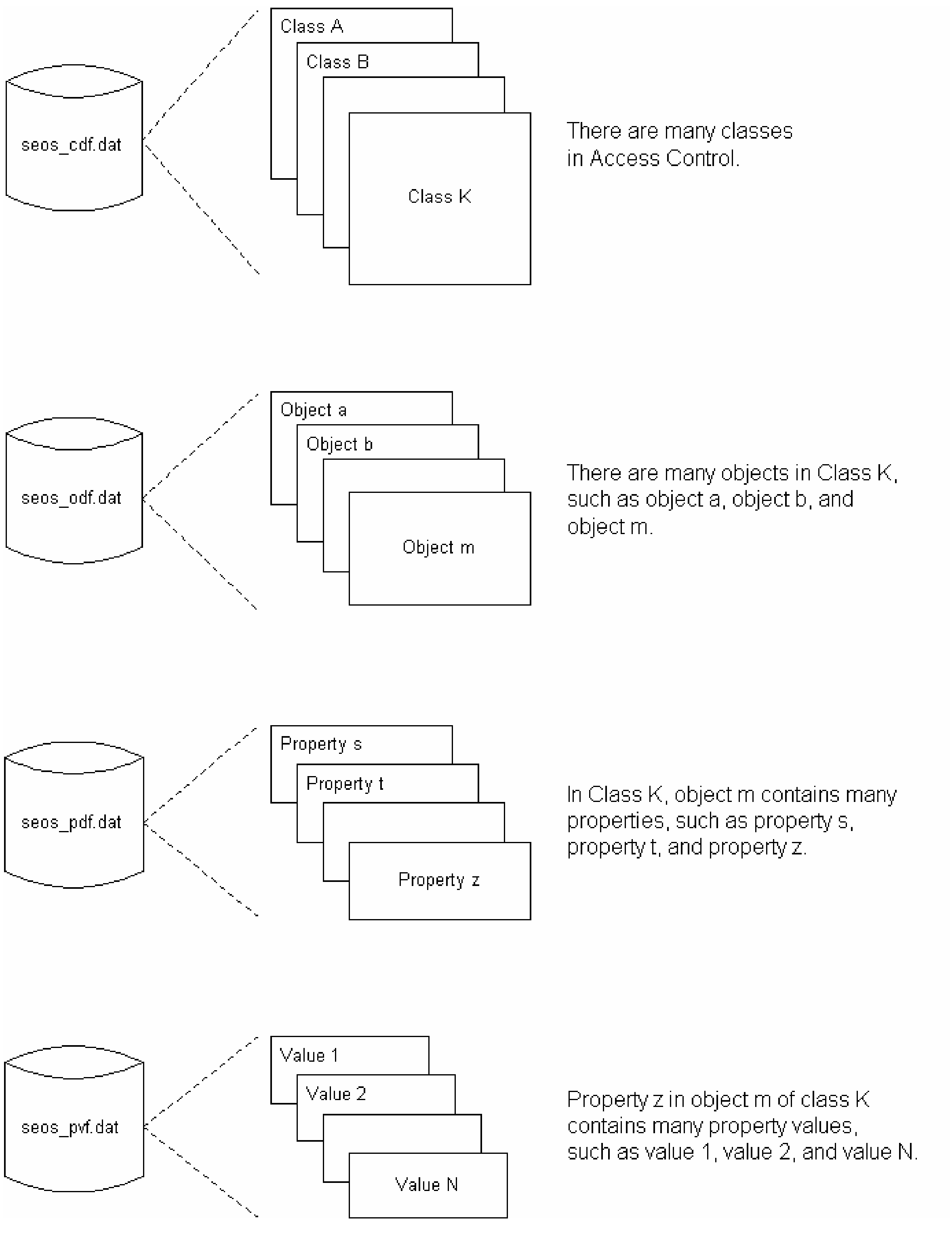
- Initializing and terminating functions, including lca_Init and lca_Terminate
- eTrust AC service request. Currently, the only function in this category is lca_ParseLine.
- Returned information analysis. Most of the LCA functions are in this category.

The eTrust AC Database

The LCA can update and query an eTrust AC local, remote, or PMDB. The Administration API can only query the database at the local station.


Programs at YourSite
Users at Your Site
Programs at Your Site
Administraion API
Access Control GUIs and selang
Language Client API
Language Client Functions
Queries
Queries and Updates
Local Access Control Database
Remote Access Control Database
Local Access Control Database
Policy Model Database

The following diagram shows the layout of an eTrust AC database:


seos_cdf.dat
Class A
Class B
Class K
There are many classes in Access Control.
seos_odf.dat
Object a
Object b
Object m
There are many objects in Class K, such as object a, object b, and object m.
seos_pdf.dat
Property s
Property t
Property z
In Class K, object m contains many properties, such as property s, property t, and property z.
seos_pvf.dat
Value 1
Value 2
Value N
Property z in object m of class K contains many property values, such as value 1, value 2, and value N.

Sample Program

The following sample program receives a command and, optionally, a list of property names as command line arguments. The program assumes that the command is a selang command, and that the property names should appear in the same format as displayed by the dbmgr -d -r p ClassName command (in older versions of eTrust AC, as displayed by the rdbdump p command).

The program invokes the lca_ParseLine function to execute the command. After execution, the results are analyzed.

If the command succeeds, the API routines that handle the query data are called, and the results of the queries are displayed. (No query information displays if the command was not a query.) If a list of properties was supplied, only data for those properties appear. If the command fails, the API routines that analyze the error are called.

For a complete description of each function in this sample program, see Language Client API Functions in this chapter.

```
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <seostype.h>
#include <langapi.h>


static void scan_entities(void);
static void print_entity_info(LCA_QENT_H entity_handle);
static void scan_props(LCA_QENT_H entity_handle);
static void print_prop_info(LCA_QPROP_H prop_handle);
static void scan_values(LCA_QPROP_H prop_handle);
static void print_prop_value(LCA_QPROP_H prop_handle,void *prop_value);


static char  *Command = NULL;
static char **Properties = NULL;
static int    NProps = 0;

static void put_commands(int argc,char **argv);

int ShowUsage(void)
{
  fprintf( stderr, "Usage:\n
                  lca_examp Command [{List-of-property-names}]\n");
  return 1;
}

int main(int argc,char **argv)
{
  char  *output = NULL;
  int   rv;
  int   n_ents;
  char  buff [1025];
     /****************** Initialization ***********************/
     /*******************************************************/

    if ( argc < 2 )
        return ShowUsage();

    rv = lca_Init("langapi",&output);
    if ( rv )
    {
        printf( "Return value: 0x%08x\n"
              "Msg: '%s'\n", rv, output );
        return 1;
    }

    put_commands(--argc,++argv);
```

```
 /************** Execution of command **********************/
 /********************************************************/

rv = lca_ParseLine(Command, &output);
/* print the command output */
printf("Command's output\n%s\n", output);

if (rv == 0)
{

 /************ Success - get the results ******************/
 /********************************************************/


 /** Get number of entities (objects) that were returned ***/
 /********************************************************/


    n_ents = lca_QEntsGetNum();
    printf("number of entities returned: %d\n",n_ents);


 /******* Get the names of the entities (objects) *********/
 /********************************************************/

    scan_entities();
}
else
{


 /****************** The command failed *****************/
 /********************************************************/

    LCA_ERR_H error_handle = NULL;


 /*********************************************************
 * For further analysis of the error, review all errors
 * (usually there is only one, but if there are warnings as
 * well as errors, there could be more than one in the list).
 ********************************************************/

    printf("Numbers of errors: %d\n\n",lca_ErrsGetNum());
    while ((error_handle = lca_ErrGetNext(error_handle))
                                                   != 0)
    {
        printf("Severity: %d, Stage:%d\n",
                                lca_ErrSeverity(error_handle),
                                lca_ErrStage   (error_handle));

     rv = lca_Err2Str (error_handle, buff, sizeof (buff)-1);
            if ( rv )
                printf ("Error message: '%s'\n", buff);
        }
      }

     /************** Terminate the use of lca_ API **************/
     /*********************************************************/

  lca_Terminate();
  return 0;
}

static void put_commands(int argc,char **argv)
{
  Command = argv[0];
  Properties = ++argv;
  NProps = --argc;
}

static void scan_entities(void)
{
  LCA_QENT_H entity_handle = NULL;

while ( (entity_handle = lca_QEntGetNext(entity_handle)) != 0 )
      print_entity_info(entity_handle);
}

/**************************************************************
Print the information for one entity, including its name, class name, and
information about its properties.
If the entity is a class, the "name" will be empty, and the "class name" full.
If the entity is an object, the "name" will be full, and the "class name" empty.
*************************************************************/


static void print_entity_info(LCA_QENT_H entity_handle)
{
  char    *name;

  if ( entity_handle == 0 )
    return;

  name = lca_QEntObjName(entity_handle);
  if ( name != 0 )
    printf("\nName: %s\t",name);
  name = lca_QEntClassName(entity_handle);
  if ( name != 0 )
    printf("Class name: %s\n",name);

  scan_props(entity_handle);

}

static void scan_props(LCA_QENT_H entity_handle)
{
  LCA_QPROP_H prop_handle = NULL;
  register int i;

    if ( NProps == 0 )
        while ( (prop_handle = lca_QPropGetNext(entity_handle,prop_handle)) != 0)
            print_prop_info(prop_handle);
    else
    for ( i = 0; i < NProps; i++ )
    {
        prop_handle = lca_QPropGetByName(entity_handle,Properties[i]);
        if ( prop_handle == 0 )
        {
            printf("Property %s does not exist for this entity.\n",
                          Properties[i]);
            continue;
        }
        print_prop_info(prop_handle);
    }
}

/************************************************************
  This function gets a property handle, and:
  1. Gets its name and prints it.
  2. Gets its value, translates it to a string, and prints it.
************************************************************/

static void print_prop_info(LCA_QPROP_H prop_handle)
{
  char *prop_name = NULL;
  char prop_type;
  unsigned short prop_size;
  unsigned int n_vals;

  if ( prop_handle == 0 )
    return;

  prop_name = lca_QPropName(prop_handle);
  if ( prop_name == 0 )
    printf("Cannot get name of property. Handle: %d\n",
                  prop_handle);

  printf("\nPropname: %s.",prop_name);

  prop_type = lca_QPropType(prop_handle);
  printf("Property type: %d; ",prop_type);

  prop_size = lca_QPropSize(prop_handle);
  printf("Property size: %d; ",prop_size);

  n_vals = lca_QPropValsNum(prop_handle);
  printf("Number of values: %d.\n",n_vals);


/* Loop over the property values, because there could be more
than one value for a property.
This happens if properties are of type list (ACLs, PACLs, User
list, Member list and so on).
*/

  scan_values(prop_handle);

}

static void scan_values(LCA_QPROP_H prop_handle)
{
  void *prop_value;
  int i;
  int n_vals;

    n_vals = lca_QPropValsNum(prop_handle);
    for ( i=0; i<n_vals; i++)
    {
      prop_value = lca_QPropValGetByIdx(prop_handle,i);
      print_prop_value(prop_handle,prop_value);
    }
}

static void print_prop_value(LCA_QPROP_H prop_handle,void
      *prop_value)
{
  char buff[1024];
  int n_vals;

  if ( prop_handle == 0 || prop_value ==0 )
    return;
  n_vals= lca_QPropVal2Str(prop_handle,prop_value,buff,sizeof(buff));
  if ( n_vals > 0 )
```

```
    printf("Value: %s\n",buff);
}
```

Language Client API Functions

The LCA includes functions in the following categories:

- Execution operations. Perform general operations, such as initializing the LCA, terminating the LCA, and executing eTrust AC commands (queries).
- Password operations. Return password rule information.
- Error handling operations. Return command error handles, and provides information about errors.
- Query operations: entity handling. Return entity handles, and provides information about found entities (objects).
- Query operations: property handling. Return property handles, and provides information about properties of found entities.
- Remote authorization operations. Perform login and query operations with the remote authorization server.

Execution Operations

The following functions control LCA operations.

lca_Init

Initializes the LCA.

lca_Terminate

Terminates the LCA.

lca_ParseLine

Performs query commands.

Password Operations

The following functions control LCA passwords.

1ca_CheckPasswordQuality

Checks new passwords for adherence to password rules.

Error Handling Operations

The following functions operate on errors.

lca_ErrsGetNum

Returns the number of messages produced by the last query command.

lca_ErrGetByIdx

Returns an indexed error handle.

lca_ErrGetFirst

Returns the handle for the first error in a command.

lca_ErrGetNext

Returns the handle for the next error in a command.

lca_ErrSeverity

Returns the severity level of an error.

lca_ErrStage

Returns information about when the error occurred.

lca_Err2Str

Converts an error record to string format and copies it to a buffer.

Query: Entity Handling Operations

The following functions operate on entities.

lca_QEntsGetNum

Returns the number of entities found by the last query command.

lca_QEntGetByIdx

Returns an entity handle for the indexed entity.

lca_QEntGetFirst

Returns an entity handle for the first entity in a list.

lca_QEntGetNext

Returns an entity handle for the next entity in a list.

lca_QEntGetByName

Returns an entity handle for the specified object in the entity list.

lca_QEntObjName

Returns the name of the object pointed to by an entity handle.

lca_QEntClassName

Returns the name of the class of the object pointed to by an entity handle.

Query: Property Handling Operations

The following functions operate on properties.

lca_QPropsGetNum

Returns the number of properties available for an entity.

lca_QPropGetByIdx

Returns a property handle for the indexed property of an entity.

lca_QPropGetFirst

Returns a property handle for the first property of an entity.

lca_QPropGetNext

Returns a property handle for the next property of an entity.

lca_QPropGetByName

Returns a property handle for the specified property of an entity.

lca_QPropName

Returns the name of a property.

lca_QPropSize

Returns the size (in bytes) of a property.

lca_QPropType

Returns the type of a property.

lca_QPropValsNum

Returns the number of values of a property.

lca_QPropValList

Returns the property value list of a property.

lca_QPropValGetByIdx

Returns an indexed property value for a single property.

lca_QPropValGetFirst

Returns the first value in a list of property values.

lca_QPropValGetNext

Returns the next value in a list of property values.

lca_QPropVal2Str

Converts a property value to string format and copies it to a buffer.

Remote Authorization Operations

The following functions control remote authorization operations.

lca_rmtauth_Init

Logs in the remote authorization server.

lca_rmtauth_CheckAccess

Determines whether a user is allowed access to a resource.

lca_Init Function

The lca_Init function initializes the LCA. You must call lca_Init before using any other LCA function. This function also assigns a global variable with the value specified in the szModuleName parameter, a character string that identifies the calling module.

This function returns zero on success and nonzero on failure. If a failure occurs, ppOutput generally points to an error message.

```
int lca_Init (const char *szModuleName, char **ppOutput);
```

szModuleName

A string that identifies the module using this API. This parameter **cannot** be null or an empty string.

ppOutput

In case of failure, a pointer to a string that contains an error message.

lca_Terminate Function

The lca_Terminate function exits the LCA. It closes any open file descriptors, frees allocated buffers, resets the module name that was set by the lca_Init function, and unloads dynamically linked libraries.

This function does not return a value.

```
void lca_Terminate (void);
```

Note: Use this function only after processing the database with the lca_ParseLine function (see its description in this chapter).

lca_ParseLine Function

The lca_ParseLine function executes the command specified by the szLine parameter. After executing the command, the lca_ParseLine function sets the ppOutput parameter to point to a character string buffer returned by eTrust AC. The caller does not need to allocate any memory. This buffer is handled internally by the LCA and freed when you call the lca_Terminate function (see its description in this chapter).

This function performs the following sequence of actions:

1. Accepts any command that you can issue from the selang command shell.
2. Builds a request block for each connected target host.
3. Accepts a set of structures with the error codes and information returned from each host.
4. Converts the values returned from the target hosts to a character string array, which it then returns through the ppOutput argument.

The lca_ParseLine function is state-sensitive; that is, it changes the behavior of the commands that follow. For example, the following sequence lists the properties of user jan from host a:

```
lca_ParseLine("hosts a", output);
lca_ParseLine("showusr jan", output);
```

A simple application that issues commands and displays the output may not need any function except lca_ParseLine. A more complex application may need more information, and, therefore, will probably need to use the other LCA functions.

This function returns zero on success and a nonzero value on error.

```
int lca_ParseLine (char *szLine, char **ppOutput);
```

szLine

A null-terminated character string containing the command to be executed.

ppOutput

A pointer to a null-terminated string containing the result returned by eTrust AC after it executes the command.

lca_ParseMBLine Function

The lca_ParseMBLine function works in the same way as the lca_ParseLine function (see page 160) except that it:

- Converts the input string from a multibyte format to UTF8 format.
- Converts the output from UTF8 format back to multibyte format.

This additional functionality support product localization and lets you input multibyte format strings.

lca_CheckPasswordQuality Function

The lca_CheckPasswordQuality function checks new passwords for users without changing them in the eTrust AC database or the native operating system through the sepass utility (for UNIX), selang, or Policy Manager. You can check passwords for adherence to eTrust AC password rules. A password is accepted or rejected based on the rules.

If the password is acceptable the function returns zero. If the password is denied, the function returns that the password is denied and gives the rule it failed.

```
int lca_CheckPasswordQuality (char *szUNAME, char *szPassword, CHECK_RSULT **res,
int domain);
```

szUName

The user requesting the password change.

szPassword

The password that the user wants to change to.

res

Specifies the handle for the result of the access request.

domain

Specifies the domain to check the password in.

Note: This variable is for future use. It is implemented only in SEOS_DOMAIN.

lca_ErrsGetNum Function

The lca_ErrsGetNum function returns the number of errors, warnings, and information messages returned by eTrust AC after executing the last command sent to it by the lca_ParseLine function (see its description in this chapter).

If there are no errors, warnings, or information messages, the function returns zero. Otherwise, it returns the total number of errors, warnings, and information messages returned.

```
int lca_ErrsGetNum (void);
```

lca_ErrGetByIdx Function

The lca_ErrGetByIdx function returns an error handle of type LCA_ERR_H for the error with index idx. The idx parameter is the index in the list of errors and messages returned by eTrust AC after it executes the last command sent by the lca_ParseLine function (see its description in this chapter).

The index ranges from 0 for the first error, to *n_errors*-1 for the last error.

You can extract information about the error or message later with this handle, by using the lca_ErrSeverity and lca_ErrStage functions. See their descriptions in this chapter.

This function returns an error handle of type LCA_ERR_H on success, and returns NULL on failure.

```
LCA_ERR_H lca_ErrGetByIdx (int idx);
```

idx

The index of the error in the error list of the last command.

lca_ErrGetFirst Function

The lca_ErrGetFirst function returns an error handle of type LCA_ERR_H for the first error that occurred in the last command sent to the lca_ParseLine function (see its description in this chapter).

You can extract information about the error or message later with this handle, by using the lca_ErrSeverity and lca_ErrStage functions. See their descriptions in this chapter.

Note: This function acts like the lca_ErrGetNext function when you invoke the latter with a NULL handle.

This function returns an error handle of type LCA_ERR_H for the first error record and returns NULL on failure.

```
LCA_ERR_H lca_ErrGetFirst (void);
```

lca_ErrGetNext Function

The lca_ErrGetNext function returns an error handle of type LCA_ERR_H for the next error that occurred in the last command sent to the lca_ParseLine function (see its description in this chapter).

You can extract information about the error or message later with this handle, by using the lca_ErrSeverity and lca_ErrStage functions. See their descriptions in this chapter.

This function returns an error handle of type LCA_ERR_H for the next error record and returns NULL on failure.

```
LCA_ERR_H lca_ErrGetNext (const LCA_ERR_H errhPrev);
```

errhPrev

The handle of the previously returned error. If the value is NULL, the first error is returned.

lca_ErrSeverity Function

The lca_ErrSeverity function returns the severity of an error, as represented by the error handle errh. The possible severity values are:

LCA_ESEV_FATAL

A fatal OS internal error (for example, memory allocation) occurred.

LCA_ESEV_ERROR

An error in the execution of a command occurred.

LCA_ESEV_WARNING

A command executed successfully, but a non-fatal error occurred.

LCA_ESEV_INFO

Not an error-the error string contains an information message or a warning.

This function returns the severity of the error and -1 on failure.

```
int lca_ErrSeverity (const LCA_ERR_H errh);
```

errh

The handle of the error returned by the lca_ErrsGetNum, lca_ErrGetByIdx, lca_ErrGetNext, or lca_ErrGetFirst function.

lca_ErrStage Function

The lca_ErrStage function returns the stage in which an error, represented by the error handle errh, occurred. The possible stage values are:

LCA_ERRUPD

An error occurred while updating the database, such as an attempt to add an existing object, or an attempt to delete a nonexistent object.

LCA_ERRAUTH

The authorization process prevented the execution of the command; for example, the user did not have the authority to execute the command.

LCA_ERRCOMM

An error occurred in communication. This usually indicates that communication with the seagent or seosd daemon failed.

LCA_ERRCONNECT

The connection to the remote host failed.

LCA_ERROTHER

An error occurred that does not belong to any of the preceding categories. Usually this indicates a syntax error or an error in the command attributes.

LCA_NOTERR

Not an error-the error string contains an information message or a warning.

This function returns the stage of the error and -1 on failure.

```
int lca_ErrStage (const LCA_ERR_H errh);
```

errh

The handle of the error returned by the lca_ErrsGetNum, lca_ErrGetByIdx, lca_ErrGetNext, or lca_ErrGetFirst function.

lca_Err2Str Function

The lca_Err2Str function translates an error record into a string, and copies the string into szBuff. If the string is longer than iBuffLen, the lca_Err2Str function truncates it.

This function returns the number of printed characters on success and -1 on failure.

```
int lca_Err2Str (const LCA_ERR_H errh, char *szBuff, int iBuffLen);
```

errh

The handle of the error returned by the lca_ErrsGetNum, lca_ErrGetByIdx, lca_ErrGetNext, or lca_ErrGetFirst function.

szBuff

A buffer into which the function copies the output string.

iBuffLen

The size of szBuff.

lca_QEntsGetNum Function

The lca_QEntsGetNum function returns the number of entities (objects) that were found by the last command sent to the lca_ParseLine function (see its description in this chapter). If the last command was not a query, lca_QEntsGetNum returns zero. Otherwise, it returns the number of entities found that match the query criteria.

This function returns the number of entities (objects) fetched by the last query. If an error occurs, the function returns -1.

```
int lca_QEntsGetNum (void);
```

lca_QEntGetByIdx Function

The lca_QEntGetByIdx function returns an entity handle for the entity with the index idx. The idx parameter is the index in the list of entities for the last command sent to the lca_ParseLine function (see its description in this chapter).

The index ranges from zero for the first entity to *n_entities*-1 for the last entity. For example, if the query returned three entities, idx can have a value of 0, 1, or 2. You can use this handle later to extract information about the entity.

For a query on a single object, such as showusr jan, the function returns an entity handle only if idx = 0; otherwise, it returns NULL. For a query on multiple objects, such as showusr jan*, it returns the entity handle specified by the idx parameter.

This function returns an entity handle of type LCA_QENT_H and returns a NULL on failure.

```
LCA_QENT_H lca_QEntGetByIdx (int idx);
```

idx

The index of the entity in the list of entities (objects) returned by the last command.

lca_QEntGetFirst Function

The lca_QEntGetFirst function returns an entity handle of type LCA_QENT_H for the first entity in the list of entities returned by the last command sent to the lca_ParseLine function (see its description in this chapter).

If you call the lca_QEntGetNext function with an attribute of NULL, you get the same results.

You can use this handle later to extract information about the entity.

This function returns only the first entity handle, regardless of how many entities the query returned. To return subsequent entity handles sequentially, use the lca_QEntGetNext function (see its description in this chapter); to return a specific entity handle, use the lca_QEntGetByIdx function (see its description in this chapter).

This function returns an entity handle of type LCA_QENT_H on success, and returns a NULL on failure.

```
LCA_QENT_H lca_QEntGetFirst (void);
```

lca_QEntGetNext Function

The lca_QEntGetNext function returns an entity handle of type LCA_QENT_H for the next entity in the list of entities for the last command sent to the lca_ParseLine function (see its description in this chapter).

If the qenthPrev parameter is NULL, the function returns the first entity in the list. You can use this handle later to extract information about the entity.

This function returns an entity handle of type LCA_QENT_H on success and a NULL on failure.

```
LCA_QENT_H lca_QEntGetNext (const LCA_QENT_H qenthPrev);
```

qenthPrev

The handle of the previous entity in the query's entity list.

lca_QEntGetByName Function

The lca_QEntGetByName function returns an entity handle of type LCA_QENT_H for one entity in the list of entities for the last command sent to the lca_ParseLine function (see its description in this chapter).

The function searches the entity list for an entity with a class name identical to szCName and an object name identical to szOName. (If szCName is empty, it searches only for a matching object name.) The function returns an entity handle for the matching entity. You can use the returned handle later to extract information about the entity.

This function returns an entity handle of type LCA_QENT_H on success and a NULL on failure.

```
LCA_QENT_H lca_QEntGetByName (const char *szCName, const char *szOName);
```

szCName

A character string with the entity's class name.

SzOName

A character string with the entity's object name.

lca_QEntObjName Function

The lca_QEntObjName function returns the name of the object to which the entity handle qenth points. The return value of the object name is empty (NULL) if the result of the query does not contain object names. For example, after executing the find command, which lists all the classes in the database, the returned string is empty because the query's result is class names and not object names. In this case, use the function lca_QEntClassName (see its description in this chapter).

This function returns a pointer to a character string that contains the object name or NULL if the object name is empty.

```
char *lca_QEntObjName (const LCA_QENT_H qenth);
```

qenth

An entity handle.

lca_QEntClassName Function

The lca_QEntClassName function returns the name of the class of the entity to which the handle qenth points. The return value of the class name is empty (NULL) if the result of the query does not contain class names. For example, the find classname command returns only the object name for the specified class.

This function returns a pointer to a character string that contains the class name on success, or NULL on failure.

```
char *lca_QEntClassName (const LCA_QENT_H qenth);
```

qenth

An entity handle.

lca_QPropsGetNum Function

The lca_QPropsGetNum function retrieves the number of queried properties for the entity to which the handle qenth points. The assumption is that the entity was retrieved by the last command executed by the function lca_ParseLine (see its description in this chapter).

The number of properties available for an object depends on the query. Exception: If the query was a find command, no properties are available because only object or class names are returned.

If the query was a showusr, showgrp , or showres command, this function counts all the properties that are set for the entity pointed to by qenth, and that can be displayed.

Note: The number of properties returned by this function is not necessarily the same as the total number of properties that exist in the database for the entity. Instead, the returned number represents the results of the last query for the entity. The entity handle is a handle that points to the data of the entity in the query, used only to access the query data using this API.

This function returns the number of properties available for the entity on success and -1 on failure.

```
int lca_QPropsGetNum (const LCA_QENT_H qenth);
```

qenth

An entity handle returned by the lca_QEntGetByIdx, lca_QEntGetNext, lca_QEntGetFirst, and lca_QEntGetByName functions.

lca_QPropGetByIdx Function

The lca_QPropGetByIdx function returns the property handle for the property whose index idx is in the property list of this entity. The index ranges from 0 for the first property, to *n_props*-1 for the last one.

You can use the returned handle later to extract data about the property, when using the lca_QPropVal2Str, lca_QPropValGetFirst, lca_QPropValGetNext, and lca_QPropValList functions.

This function returns a property handle for the requested property. If no property with the specified index exists for this entity, the function returns NULL.

```
LCA_QPROP_H lca_QPropGetByIdx (const LCA_QENT_H qenth, int idx);
```

qenth

An entity handle that identifies an entity in the query.

idx

The index of the requested property in the entity's property list.

lca_QPropGetFirst Function

The lca_QPropGetFirst function returns a property handle for the first property of the entity pointed to by the qenth parameter. You can use the returned handle later to extract data about the property, when using the lca_QPropVal2Str, lca_QPropValGetFirst, lca_QPropValGetNext, and lca_QPropValList functions.

This function returns a property handle. If there are no properties available in the query result of this entity, the function returns NULL.

```
LCA_QPROP_H lca_QPropGetFirst (const LCA_QENT_H qenth);
```

qenth

An entity handle that identifies an entity in the query.

lca_QPropGetNext Function

The lca_QPropGetNext function returns the property handle of the property after the one in the qprophPrev parameter, for the entity pointed to by the qenth parameter. If the qprophPrev function is NULL, it returns the first property in the list.

You can use the returned handle later to extract data about the property, when using the lca_QPropVal2Str, lca_QPropValGetFirst, lca_QPropValGetNext, and lca_QPropValList functions.

This function returns a property handle. If there are no more properties available in the query result of this entity, the function returns NULL.

```
LCA_QPROP_H lca_QPropGetNext (const LCA_QENT_H qenth, \
const LCA_QPROP_H qprophPrev);
```

qenth

An entity handle that identifies an entity in the query.

qprophPrev

Property handle of the last retrieved property.

lca_QPropGetByName Function

The lca_QPropGetByName function returns a property handle-for the property whose name is in the szPName parameter-of the entity to which the qenth parameter points. You can use the returned handle later to extract data about the property, when using the lca_QPropVal2Str, lca_QPropValGetFirst, lca_QPropValGetNext, and lca_QPropValList functions.

This function returns a property handle. If there are no properties with this name, the function returns NULL.

```
LCA_QPROP_H lca_QPropGetByName (const LCA_QENT_H qenth const char szPName);
```

qenth

An entity handle that identifies an entity in the query.

szPName

A character string representing a property name.

lca_QPropName Function

The lca_QPropName function receives a property handle that identifies one property of an entity, and returns the property's name. This function belongs to a group of functions that retrieve information about a single property. The property name is the name defined in the database or, for UNIX queries, the symbolic name eTrust AC gives to UNIX properties.

This function returns a character string representing the property name on success and NULL on failure.

```
char *lca_QPropName (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropSize Function

The lca_QPropSize function receives a property handle that identifies one property of an entity, and returns the property's size in bytes. This function belongs to a group of functions that retrieve information about a single property. The size is returned for a single value of the property. If it is a list property-such as an access control list (ACL)-the size of a single element is returned. To determine the size of the whole list, multiply the size of a single value by the number of values in the list. You can retrieve the number of values in the list by using the lca_QPropValsNum function.

This function returns the size of the property on success and -1 on failure.

```
unsigned short lca_QPropSize (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropType Function

The lca_QPropType function receives a property handle that identifies one property of an entity and returns the property's type. This function belongs to a group of functions that retrieve information about a single property. The possible property types are listed in the seostype.h header file.

This function returns a single character representing the property type on success and -1 on failure.

```
char lca_QPropType (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropValsNum Function

The lca_QPropValsNum function receives a property handle identifying one property of an entity and returns the number of values for the property. This function belongs to a group of functions that retrieve information about a single property. For most properties, the number is always one. However, there are properties composed of lists that may have more than one value. Some examples of properties composed of lists are ACL, PACL, INetACL, UserList, and GroupList.

This function returns the number of values for the property on success. If the function fails, it returns -l. If the fetch from the database failed, the function returns no value for the property.

```
unsigned int lca_QPropValsNum (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropValList Function

The lca_QPropValList function returns the property value list for a single property identified by the qproph parameter. The returned value is an array of pointers to the values. If the property has a single value, only the first pointer is valid. To determine the number of valid pointers in the list, call the lca_QPropValsNum function (see its description in this chapter).

Note: You can find the format of the property values, according to each value's type, in the seadmapi API documentation (see the chapter "Administration API").

This function returns a pointer to an array of pointers to the property values on success and NULL on failure.

```
void **lca_QPropValList (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropValGetByIdx Function

The lca_QPropValGetByIdx function returns a single property value for a single property identified by the qproph parameter. This function identifies the value by its index in the value list for the property. The index ranges from 0 to one less than the value returned by the lca_QPropValsNum function (see its description in this chapter).

To determine the number of values in the list, call lca_QPropValsNum. If necessary, use the lca_QPropVal2Str function to convert this value to a string.

Note: You can find the format of the property values, according to each value's type, in the seadmapi API documentation (see the chapter "Administration API").

This function returns a pointer to a single property value on success and NULL on failure.

```
void *lca_QPropValGetByIdx (const LCA_QPROP_H qproph, int idx);
```

qproph

A property handle.

idx

An index to a value in the property (in case of a list property).

lca_QPropValGetFirst Function

The lca_QPropValGetFirst function returns the first value in the list of values for the property identified by the parameter qproph. If necessary, use the lca_QPropVal2Str function to convert the value to a string.

Note: You can find the format of the property values, according to each value's type, in the seadmapi API documentation (see the chapter “Administration API”).

This function returns a pointer to a single property value on success and a NULL on failure.

```
void *lca_QPropValGetFirst (const LCA_QPROP_H qproph);
```

qproph

A property handle.

lca_QPropValGetNext Function

The lca_QPropValGetNext function returns the next property value in the list of values for the property identified by the parameter qproph. The pPrevVal parameter identifies the previous value in the list. The function returns the value following pPrevVal. If pPrevVal is NULL, the function returns the first value in the list.

You can convert the value returned to a string later, using the function lca_QPropVal2Str.

Note: You can find the format of the property values, according to each value's type, in the seadmapi API documentation (see chapter “Administration API”).

This function returns a pointer to a single property value on success and a NULL on failure.

```
void *lca_QPropValGetNext (const LCA_QPROP_H qproph, const void *pPrevVal);
```

qproph

A property handle.

pPrevVal

The previous value in the list.

lca_QPropVal2Str Function

The lca_QPropVal2Str function translates a property value into a string and copies the string into szBuff. If the string is longer than iBuffLen, the lca_QPRopVal2Str function truncates it.

The property handle is needed to analyze the value in the pVal parameter correctly, based on its type and size.

The function translates a single value each time it is called. You must call the function lca_QPRopVal2Str separately for each property value in a property list such as ACL.

This function returns the number of printed characters on success and -1 on failure.

```
int lca_QPropVal2Str (const LCA_QPROP_H qproph, \
const void *pVal, char *szBuff, int iBuffLen);
```

qproph

A property handle.

pVal

A pointer to the property value to be translated.

szBuff

A buffer into which the function copies the output string.

iBuffLen

The size of szBuff.

lca_rmtauth_Init Function

The lca_rmtauth_Init function attempts to log in to the remote authorization server specified by the szHost parameter. This establishes a connection to the server checks whether the password passed by the szPasswd parameter matches the password of the user running the application. If it matches, the login process continues.

This function returns zero if login is successful, and a nonzero on failure. If login fails, the ppOutput parameter points to an error message.

Note: For a complete discussion of the Remote Authorization API, see the chapter “Authorization and Authentication API.”

```
int lca_rmtauth_Init (const char *szModuleName, \
char **ppOutput, char *szHost, char *szPasswd);
```

szModuleName

The name of the application. This parameter **cannot** be null or an empty string.

ppOutput

In case of failure, a pointer to an error message string.

szHost

The name of the server to connect to.

szPasswd

The password of the user accessing the server.

lca_rmtauth_CheckAccess Function

The lca_rmtauth_CheckAccess function queries the eTrust AC remote authorization server to determine whether the user-specified by szAccessor-is permitted to access the specified resource (szOName) when using the specified access type (szAccs). The function sends the request to the remote authorization server, which then performs the check using the SEOSROUTE_RequestAuth function.

This function returns zero if login is successful and a nonzero value on failure.

Note: For a complete discussion of the Remote Authorization API, see the chapter "Authorization and Authentication API."

```
int lca_rmtauth_CheckAccess (const char *szAccessor, \
char *szClass, char *szOName, char *szAccs, API_AUTH_RES *result);
```

szAccessor

The name of the accessor (user) requesting authorization.

szClass

The resource whose access authorization is to be checked.

szOName

The resource object to which the user is requesting access.

szAccs

The type of access requested. The valid access types are listed in the seostype.h header file.

result

The server's result of the access request.

Chapter 6: Administration API

Programming Guide

eTrust AC uses an object-oriented database. The database includes definitions for classes, objects, and specific class definition of data members. It is essential to understand the database architecture before using the Administration API, because the API includes functions for reading and modifying the information in the database.

The *Administrator Guide* explains some of the details about the database layout. We recommend reading that guide in addition to this chapter.

Database Organization

The information in the database is organized into classes. A class's definition includes information that is common to all records, or objects, of that class. Records that belong to the same class have a similar meaning. For example, every record in the USER class represents a user; every record in the GROUP class represents a group of users; and every record in the TERMINAL class represents a terminal from which users can access the current host. Every class contains a properties definition table that includes a list of properties that you can assign to records belonging to the class.

A record is a single entity that represents an instance of its class. For example, a record in the USER class represents an individual user. Each class contains properties, or fields, that are specific to the class. Information is stored in a record by assigning values to its properties. The definition of a property includes information on the layout of the data and attributes that define how the data is stored in the database. You can assign values to every property of a record. The definition of the record's class determines which properties you can assign to the record and what values you can assign to each property.

The structure of the database is best described by an example. Consider the USER class. Every record in the USER class represents a user of the system. The properties definition table of the USER class contains a list of properties that you can assign to user records. Some of the properties in the list are FULL_NAME, ORGANIZATION, and GROUPS. You can assign values for these properties to every user represented by a record in the database. For example, the FULL_NAME property stores the user's full name; the ORGANIZATION property stores information about the organization the user belongs to; and the GROUPS property contains a list of the groups to which the user belongs. The properties of a record are similar to fields in a database record. The format of a property can vary depending on the property definition. In this example, the FULL_NAME and ORGANIZATION properties have only a single value, whereas the GROUPS property is a list that consists of a variable number of repetitive elements. A user can belong to an unlimited number of groups.

Each class in the database has a name and an ID associated with the name. The database engine uses the class ID internally to achieve better performance and smaller database file sizes. The class IDs do not have any meaning beyond their internal use and may differ between different databases. Use class names; do not use class IDs.

Each property of a class in the database has a name and an ID associated with the property name. The database engine uses the property ID internally to achieve better performance and smaller database file sizes. Note that every class has its own properties definition table; thus, a property name can appear in more than one class and the property may have different attributes in different classes. A property is identified either by its name and the name of the class in which it resides or by its unique property ID.

Every record in the database has a name and belongs to a class. Records in different classes can have the same name. Each record in the database is associated with a record ID. The record ID is a 32-bit number that is unique to each record in the database. eTrust AC uses the record ID internally. When referring to a record, you can use the record name or the record ID.

Every record in the database can have values assigned to its properties. eTrust AC sets some properties automatically, and the user sets others explicitly. The security administrator or a delegated responsible person sets most properties by using the tools included with eTrust AC.

For detailed information about the properties supported by each class, see the *Reference* Guide.

Database Layout

The database consists of the following data files:

seos_cdf.dat

The class description file contains the class definition table.

seos_odf.dat

The objects description file contains the records definition table.

seos_pdf.dat

The properties description file contains the properties definition table.

seos_pvf.dat

The properties value file contains the values assigned to every eTrust AC property.

The data files also have indexing files that are not mentioned here because they are transparent to the Administration API.

Class Description File

The class description file stores information on all classes defined to eTrust AC. The information stored therein includes the name of the class, the class ID, and other flags used internally by eTrust AC. The class information is stored in a structure called SEOSDB_CDF.

Properties Description File

The properties description file stores information on each property defined to eTrust AC. The property information includes:

- Property ID
- Name of the property
- Class ID of the class in which the property is defined
- The property's data type-string, integer, structure, and so forth
- The size, in bytes, required to store a single value of the property

Some properties are defined as single value, and others are defined as a list value. The properties description information is stored in a structure called SEOSDB_PDF.

Objects Description File

The objects description file stores basic information on each record defined to eTrust AC. The following data is stored in the objects description file:

- Class ID of the class in which the record is defined
- Name of the record
- The record's unique internal database ID, also known as the object ID

The objects description information is stored in a structure called SEOSDB_ODF.

Properties Values File

The properties values file contains the values assigned to every property of every record defined in the database. Each entry consists of:

- Class ID
- Property ID
- Record ID
- The data assigned to the property

Information used for integrity checking is also stored in the properties values file; however, this information cannot be accessed using the Administration API.

Database Lists

This section describes the various types of lists that exist in the database.

User to Group Connections

Both user records and group records contain data that define the connections of users to groups.

The user record contains a list of groups to which the user belongs. The following information is stored in the user record:

- The date on which the connection was created
- The user or group that owns the connection
- Group attributes, if any, assigned to the user

The group record contains a list of users who are connected to the group. The list contains only the record IDs of the users.

When connecting a user to a group, the list of groups connected to the user, and the list of users connected to the group, must be updated. When a user is connected to a group, eTrust AC automatically updates both lists. If a user is subsequently deleted from the database, the user cannot be deleted from every group record containing the user's ID. Thus, some group records may contain user IDs of users who no longer exist in the database. eTrust AC generates new object IDs in a manner that ensures that an ID cannot be assigned to an object more than once in the lifetime of the database. The unused user IDs in the group records do not pose a security threat.

Connections of Resources to Resource Groups

Like a user-group connection, the connection of a resource to a resource group is stored in both the resource record and the resource group record. The resource record contains a list of record IDs that identifies the resource groups to which the resource is connected. The resource group record contains a list of resource IDs that identifies the resources that are connected to the resource group. The resource and resource group records are automatically updated whenever a resource is connected to a resource group.

ACL Entries

The access control list (ACL) is a list of zero or more entries in a resource record. Each entry in the ACL defines the access an accessor object in the database has to the resource. Each ACL entry consists of:

- Record ID of the accessor-usually a user ID or a group ID
- Access authority assigned to the accessor

This entry determines what the accessor is allowed to do to the resource represented by the resource record.

eTrust AC also provides program access control lists (PACLs), also known as conditional ACLs, which are similar to regular ACLs. In addition to the record ID of the accessor and the accessor's level of authority, PACL entries consist of a PROGRAM record ID.

By convention, eTrust AC does not assign the object ID of zero to any object. In ACLs and PACLs, an object ID of zero represents user (*)-that is, all eTrust AC defined users.

Note: For more information about ACLs, see the authorize command in the *Reference Guide*.

Understanding ACEE

eTrust AC assigns an accessor environment element (ACEE) to each user when the user logs into the system. The ACEE is a data structure containing the user's credentials and definitions of various security parameters. Every process created by the login process inherits the parent process's ACEE. The ACEE is maintained even if the process substitutes user by executing the system's su utility or the sesu utility.

Each ACEE has a handle that uniquely describes the process's credentials and other information at any point in time. The ACEE and its associated handle exist until the login session that created them terminates.

The Administration API and all eTrust AC authorization processes use the ACEE handle to identify and describe the user making the request.

The Administration API includes functions that fetch a user's ACEE or ACEE handle. In UNIX, the information obtained by these functions can be viewed using the sewhoami utility with the appropriate options. For more information, see the sewhoami utility in the *Utilities Guide*.

Scope Limitations of the API

The Administration API uses a simpler security scope method than the eTrust AC language interpreter, so as not to adversely affect performance.

The Administration API uses the attributes set in the users' USER records, but ignores other privileges that use ownership, group attributes, and the ADMIN class. This means that users cannot perform some operations using the Administration API that they can perform by using selang, selangx, the eTrust AC Administrator (seadm), or Policy Manager. For example, in selang a user can display or update an object that the user owns. The Administration API, however, does not allow an owner of an object to update it unless the owner also has the ADMIN attribute.

Conventions

The Administration API uses the following conventions:

- All function names start with seadmapi_.
- Unless otherwise documented, all functions return an int value representing the result code.
- A return code of zero always denotes a successful operation.
- Variables are always required. If a variable is a pointer, a pointer must be supplied. A pointer can be NULL only where specified.
- The library functions assign values to the C global variable errno, which is also used to return error codes.
- The names of parameters that are NULL character terminated strings (ASCII-Z or C-Style strings) are preceded by the letters sz.
- Many functions use pre-fetched information, such as class descriptions, property descriptions, and object descriptions, to speed up the operation. Use these pre-fetched descriptions wherever possible to reduce the load on the station.

Header Files

To use this API, you are required to include in your source code the header files with prototypes and structure definitions. All prototypes are in the seadmapi.h file, while most of the data types are in other headers. The seostype.h header file provides structure definitions of all data stored in the database. The structure definitions of auditing and error logging records are located in the header file selogtype.h.

Libraries

This section discusses the libraries that must be used with this API.

In UNIX

seadmapi consists of a single library file, seadmapi.a, that you should link with every compiled source file that uses this API.

eTrust AC includes a shared library version of this API called libseadmapi.*xx*, where *xx* is the standard operating system convention (usually so or sl) for shared library names. Before executing programs that use the shared library, such as sample_TermOwn.c, check that an environment variable points to the path of the shared library. To point an environment to the shared path, enter:

```
setenv LD_LIBRARY_PATH /opt/CA/eTrustAccessControl/lib
```

In Windows

To compile and link your program with seadmapi functions, make sure you include the static library seadmapi.lib in your link path. This library is usually found in eTrustACDir\lib (where *eTrustACDir* is the directory you installed eTrust AC in, by deafult Program Files\CA\eTrustAccessControl).

Before executing programs that use this library, check that an environment variable points to the path of the library.

Compiling and Linking with seadmapi

There are no special flags required to compile with seadmapi. Linking, on the other hand, may require additional settings. Unfortunately, these flags are machine and operating system dependent. Use the makefiles provided by the samples of this API, and look at those samples for up-to-date information.

Programming Notes

All functions provided by this API are thread-safe. If a function is not thread-safe, then the Notes section of the function specifies that fact.

Note: You must call the seadmapi_init or seadmapi_IsSeOSSyscallLoaded function before calling any other function in the seadmapi library.

Function Library

The Administration API includes functions that are categorized as follows:

- Class operations. Retrieve list of classes and get class characteristics information.
- Properties operations. Retrieve properties of a class and get properties characteristics.
- Objects operations. Retrieve objects and get object characteristics.
- Values operations. Retrieve values, set values, or update values in the database.
- Query operations. Perform functions connected to queries.
- Log files interface. Provide means to add audit and error log records.
- Console operations. Perform functions connected to console operations.
- Miscellaneous operations. Some generic operations, such as getting commonly required information from seosd or setting process-specific data.

Class Operations Functions

The following functions operate on the eTrust AC classes:

seadmapi_ClassGetEqual

Retrieves a specific class from the database.

seadmapi_ClassGetFirst

Retrieves first class from the database.

seadmapi_ClassGetNext

Retrieves next class from the database.

Property Operations

The following functions operate on properties:

seadmapi_PropGetEqual

Retrieves description of a specific property.

seadmapi_PropGetFirstInClass

Retrieves first property description of a class.

seadmapi_PropGetNextInClass

Retrieves next property description of a class.

Object Operations

The following functions operate on objects:

seadmapi_FreeObjList

Frees the list of objects retrieved by ObjInClassList.

seadmapi_ObjGetEqual

Retrieves information on a specific object.

seadmapi_ObjGetFirstInClass

Retrieves information of the first object in a class.

seadmapi_ObjGetGreaterEqual

Retrieves information on an object whose object ID is greater than or equal to the specified object ID.

seadmapi_ObjGetNextInClass

Retrieves information on next object in a class.

seadmapi_ObjInClassList

Retrieves a list of objects in a specified class.

Value Operations

The following functions operate on values:

seadmapi_FetchListPropVal

Gets the values for a list type property.

seadmapi_FetchSinglePropVal

Gets the values for a single value property.

seadmapi_FreeListPropVal

Free the list of values from FetchListPropVal.

seadmapi_SetSinglePropVal

Sets the value of one property value type.

Query Operations

The following functions perform queries:

seadmapi_GetEntity

Retrieves an entire object and its properties values, using the previously initialized entity ruler.

seadmapi_GetExEntity

Retrieves an entire object and its properties values, including the object and class names, using the previously initialized entity ruler.

seadmapi_GetGraceInfo

Retrieves grace information about the user.

seadmapi_InitEntityRuler

Initializes an entity query buffer used for GetEntity and GetExEntity operations.

seadmapi_KillExEntityMem

Frees memory allocated for entity style query by the InitEntityRuler function.

seadmapi_KillPDFList

Frees the list of properties descriptors allocated by the MakePDFList function.

seadmapi_MakePDFList

Creates a property descriptors list from a list of properties names.

seadmapi_OidToName

Translates an object ID to object name.

Log Files Interface

The following functions operate on the log files:

seadmapi_SendAdminAudit

Submits an ADMIN audit record.

seadmapi_SendAuditRecord

Provides interface to submit audit record.

seadmapi_SendCwsAudit

Submits a connect-with-service resource audit record.

seadmapi_SendErrorLog

Submits a note to the error log.

seadmapi_SendGenrAudit

Submits a general-resource audit record.

seadmapi_SendInetAudit

Submits a TCP/IP audit record.

seadmapi_SendLoginAudit

Submits a login audit record.

seadmapi_SendShutdownAudit

Submits an audit record of shutdown.

seadmapi_SendStartupAudit

Submits an audit record of startup.

seadmapi_SendUserAudit

Submits a user audit record.

seadmapi_SendWatchdogAudit

Submits a watchdog audit record.

seadmapi_SendNfAdminAudit

Submits an ADMIN notification record.

seadmapi_SendNfCwsAudit

Submits a connect-with-service resource notification record.

seadmapi_SendNfGenrAudit

Submits a general-resource notification record.

seadmapi_SendNfInetAudit

Submits a TCP/IP notification record.

seadmapi_SendNfLoginAudit

Submits a login notification record.

seadmapi_SendNfShutdownAudit

Submits a notification record of shutdown.

seadmapi_SendNfStartupAudit

Submits a notification record of startup.

seadmapi_SendNfUserAudit

Submits a user notification record.

seadmapi_SendNfWatchdogAudit

Submits a watchdog notification record.

Console Operations

The following functions provide console operations:

seadmapi_consAllLoginDisable

Disables all login to system.

seadmapi_consAllLoginEnable

Enables all login to system.

seadmapi_consAllLoginGetStatus

Gets status of global-login control.

seadmapi_consMessageSend

Sends a message to eTrust AC trace.

seadmapi_consRefreshIPAddresses

Refreshes host name to IP address resolution.

seadmapi_consRunTimeStatisticsGet

Gets runtime statistics information.

seadmapi_consShutdown

Shuts down eTrust AC.

seadmapi_consTraceClear

Clears the trace file.

seadmapi_consTraceDisable

Disables eTrust AC trace.

seadmapi_consTraceEnable

Enables eTrust AC trace.

seadmapi_consTraceGetStatus

Returns status of eTrust AC trace.

seadmapi_consTraceToggle

Toggles eTrust AC trace.

seadmapi_consUidLoginDisable

Disables login for user ID.

seadmapi_consUidLoginEnable

Enables login for user ID.

seadmapi_consUidLoginGetStatus

Gets UID concurrent login status.

Miscellaneous Operations

The following functions perform functions that do not fall into any of the previous categories:

seadmapi_FreeAceeMemory

Frees memory allocated by the seadmapi_GetACEE function.

seadmapi_GetACEE

Retrieves the current process user's ACEE.

seadmapi_GetMessage

Retrieves an error string from a given error code, using the eTrust AC message file.

seadmapi_GetObjType

Retrieves information on the user type of the current process.

seadmapi_init

Initializes the communication channel with eTrust AC.

seadmapi_IsSeOSSyscallLoaded

Determines if eTrust AC system call is loaded.

seadmapi_ProcessControl

Provides control over current process.

seadmapi_WhoAmI

Retrieves information on current process

seadmapi_WhoIs

Retrieves attribute information about the user.

sepass_ReplacePassword

Replaces the user password with a new password.

seadmapi_ClassGet Functions

These functions retrieve information on a class that is defined in the database.

- The seadmapi_ClassGetFirst function retrieves information on the first class defined in the database.
- The seadmapi_ClassGetNext function retrieves information on the next class that is defined in the database. This function uses the class ID from the previous call to the seadmapi_ClassGetNext function or, if this is the first time the seadmapi_ClassGetNext function is being called, the class ID is obtained from the seadmapi_ClassGetFirst function. The classes are scanned in the order of their class names.
- The seadmapi_ClassGetEqual function retrieves information about a specific class. The class is identified by its class name or class ID.

To scan all the classes in the database, first call the seadmapi_ClassGetFirst function, and then call the seadmapi_ClassGetNext function for each subsequent class.

These functions can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns zero; if it fails, it returns an error code.

```
int seadmapi_ClassGetEqual(const char *szClass, \
                     SEOS_CID   cid, \
                     SEOSDB_CDF *p_seclass);
int seadmapi_ClassGetFirst(SEOSDB_CDF *p_seclass);
int seadmapi_ClassGetNext(SEOSDB_CDF *p_seclass);
```

szClass

The name of the class whose information is to be retrieved. If a class ID is specified in the cid parameter, set this parameter to NULL.

cid

The class ID of the class whose information is to be retrieved. If a class name is specified for the szClass parameter, set this parameter to -1.

p_seclass

A pointer to the structure that is to hold the information retrieved by the function. For seadmapi_ClassGetNext, the data structure must contain the values from a previous call to the ClassGetNext function or the ClassGetFirst function.

Example

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 8.0
File          : sample_ListClass.c
Purpose          : Sample seadmapi: List class names.
=============================================================

Copyright :
Copyright 2004 Computer Associates International, Inc.
=============================================================*/
#include <ctype.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <seadmapi.h>
int main (void)
{ SEOSDB_CDF     cdf;
  int            rv;
/* -------------------------------------------------------- */
/*  Get the first class from the database.                  */
/* -------------------------------------------------------- */
  rv = seadmapi_ClassGetFirst(&cdf);
  if ( rv )
     { printf("seadmapi_ClassGetFirst returned 0x%04x\n", rv );
       return 1;
     }
/* -------------------------------------------------------- */
/*  If successful, continue looping for all the classes.   */
/* -------------------------------------------------------- */
  while (!rv)
    { printf( "%s\n", cdf.szCName );
      rv = seadmapi_ClassGetNext(&cdf);
    }
  return 0;
}
```

seadmapi_PropGet Functions

These functions retrieve information on one or more properties defined in the database.

- The seadmapi_PropGetFirstInClass function retrieves information about the first property defined for the specified class.
- The seadmapi_PropGetNextInClass function retrieves information about the next property that is defined in the database for the class. This function uses the property ID from the previous call to the seadmapi_PropGetNextInClass function or, if this is the first time the seadmapi_PropGetNextInClass function is called, the property ID is obtained from the seadmapi_PropGetFirstInClass function. The properties are scanned alphabetically by their property names.
- The seadmapi_PropGetEqual function retrieves information about a specific property, identified by its property name or property ID.

To scan all the properties in a specific class, first call the seadmapi_PropGetFirstInClass function, and then call the seadmapi_PropGetNextInClass function for each subsequent property.

These functions can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions. Any process can issue a seadmapi_PropGetEqual request on any property.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_PropGetEqual(const char *szClass,
                    SEOSDB_CDF *p_seclass,
                    const char *szProp,
                    SEOS_PID   pid,
                    SEOSDB_PDF *p_seprop);
int seadmapi_PropGetFirstInClass(const char *szClass,
                    SEOSDB_CDF *p_seclass,
                    SEOSDB_PDF *p_seprop);
int seadmapi_PropGetNextInClass(SEOSDB_PDF *p_seprop);
```

szClass

The class name. When specifying a class description instead of a class name, set this parameter to NULL.

p_seclass

The class description. When specifying a class name instead of a class descriptor, set this parameter to NULL. Note that when this parameter is NULL, szClass must not be NULL.

szProp

The property name. When specifying a property ID instead of a property name, set this parameter to NULL.

pid

The property ID. When specifying a property name instead of a property ID, set this parameter to -1.

p_seprop

A pointer to the data structure that is to hold the information retrieved by the function.

Example

The following example demonstrates the use of seadmapi_PropGetFirstInClass and seadmapi_PropGetNextInClass to retrieve the values of a property in a class.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                                   Version: 8.0
File          : sample_ListProp.c
Purpose          : List properties of a specific class.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])
{ SEOSDB_CDF    cdf;                     /* Class Description    */
  SEOSDB_PDF    prop;                    /* Property Description */
  char          Class[CNAME_SIZE+1];
  unsigned      props_cnt = 0;
  int           rv;
  if ( argc < 2 )
     { fprintf(stderr, "Required parameter (class name) is
                           missing.\n");
       return 1;
     }
  /* ------------------------------------------------------- */
  /* Set class name by specified parameter.                     */
  /* ------------------------------------------------------- */
  strcpy(Class, argv[1]);
  /* ------------------------------------------------------- */
  /* Clear property descriptor.                                  */
  /* ------------------------------------------------------- */
  memset( &prop, 0, sizeof(prop) );
  /* ------------------------------------------------------- */
  /* Check if class exists by getting the class descriptor.    */
  /* ------------------------------------------------------- */
  rv = seadmapi_ClassGetEqual(Class, 0, &cdf);
  if (rv) return ErrorMessage(rv);
  /* ------------------------------------------------------- */
  /* Set the class ID in the property descriptor.               */
  /* ------------------------------------------------------- */
  prop.sCId = cdf.sCId;
  /* ------------------------------------------------------- */
  /* Loop for all the properties in the class.
 */
  /* Check for rv 0 or 1  to find all                              */
  /* the properties that are equal to or greater than       */
  /* the supplied property, which is 0.                         */
  /* ------------------------------------------------------- */
  rv = seadmapi_PropGetFirstInClass( NULL, &cdf, &prop );
  while ( (rv == 0) || (rv == 1) )
     { if ( prop.sCId == cdf.sCId )
                      { props_cnt++;
                        printf("%s %s\n", Class, prop.szPName);
                      }
       rv = seadmapi_PropGetNextInClass( &prop );
     }
  if ( props_cnt == 0 )
     printf("Class %s, does not contain this property.\n",
Class);
  return 0;
}
/* -------------------------------------------------------- */
/* Display error message.                                     */
/* -------------------------------------------------------- */
static int ErrorMessage( int rv )
{
  char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

seadmapi_ObjGet Functions

These functions retrieve information on an object (record) in the database.

- The seadmapi_ObjGetFirstInClass function retrieves information about the first object defined to a class.
- The seadmapi_ObjGetNextInClass function retrieves information about the next object defined in the class. The class ID is obtained by incrementing the current class ID in the p_seobj structure by 1. The p_seobj structure must have been set by a previous call to seadmapi_ObjGetFirstInClass or seadmapi_ObjGetNextInClass.

  To scan all the objects in a class sequentially, first call seadmapi_ObjGetFirstInClass and then seadmapi_ObjGetNextInClass for each subsequent object.
- The seadmapi_ObjGetEqual function retrieves information about a specific object. The object is identified by its object name or object ID.
- The seadmapi_ObjGetGreaterEqual function retrieves information about the object whose object ID or object name is greater than or equal to the specified value.

These functions can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

The seadmapi_ObjGetGreaterEqual function returns one of the following values:

0

The function retrieved information on the object whose object ID is equal to the object ID of the specified object.

1

The function retrieved information on the object whose object ID is greater than the object ID of the specified object.

other

The function failed.

The remaining functions return 0 on success and an error code on failure.

```
int seadmapi_ObjGetEqual(const char *szClass,
                    SEOSDB_CDF *p_seclass,
                    const char *szObj,
                    SEOS_OID
                    oid,
                    SEOSDB_ODF *p_seobj);
int seadmapi_ObjGetFirstInClass(const char *szClass,
                      SEOSDB_CDF *p_seclass,
                     SEOSDB_ODF *p_seobj);
int seadmapi_ObjGetNextInClass(SEOSDB_ODF *p_seobj);
int seadmapi_ObjGetGreaterEqual(const char *szClass,
                       SEOSDB_CDF *p_seclass,
                    const char *szObj,
                    SEOS_OID   oid,
                    EOSDB_ODF *p_seobj);
```

szClass

The name of the class to which the object belongs. If the class is specified using the p_seclass parameter, set this parameter to NULL.

p_seclass

A pointer to a structure containing the class descriptor. If the szClass parameter is specified, set this parameter to NULL.

szObj

The name of the object whose value is to be fetched. If an object ID is specified instead of an object name, set this parameter to NULL.

oid

The object ID of the object whose information is to be retrieved. When specifying an object name instead of an object ID, set this parameter to -2.

p_seobj

A pointer to the structure that is to hold the information retrieved by the function.

Example

The following example demonstrates the use of seadmi_ObjGetFirstInClass and seadmi_ObjGetNextInClass to retrieve all the objects in a specific class.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                                 Version: 8.0
File          : sample_ListObjs.c
Purpose          : Display the objects in a class.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])
{ SEOSDB_ODF    odf;                       /* Current ODF in loop */
  SEOSDB_CDF    cdf;                       /* Class Description   */
  char          Class[CNAME_SIZE+1];
  unsigned      ents = 0;
  int           rv;
  if ( argc < 2 )
     { fprintf(stderr, "Required parameter missing.\n");
       fprintf(stderr, "Usage: '%s CLASS_NAME'\n", argv[0]);
       return 1;
     }
  /* ------------------------------------------------------- */
  /* Set class name by specified parameter.                       */
  /* ------------------------------------------------------- */
  strcpy(Class, argv[1]);
  /* ------------------------------------------------------- */
  /* Clear object descriptor.                                     */
  /* ------------------------------------------------------- */
  memset( &odf, 0, sizeof(odf) );
  /* ------------------------------------------------------ */
  /* Check if class exists by getting the class descriptor.  */
  /* ------------------------------------------------------- */
  rv = seadmapi_ClassGetEqual(Class, 0, &cdf);
  if (rv) return ErrorMessage(rv);

  /* ------------------------------------------------------- */
  /* Set the class ID in the object descriptor.                   */
  /* ------------------------------------------------------- */
  odf.sCId = cdf.sCId;
  /* ------------------------------------------------------- */
  /* Loop for all the objects in the class.                   */
  /* ------------------------------------------------------- */
  rv = seadmapi_ObjGetFirstInClass( NULL, &cdf, &odf );
  while ( (rv == 0) || (rv == 1) )
     { if ( odf.sCId == cdf.sCId )
                     { ents++;
                       printf("%s %s\n", Class, odf.szOName);
     }
       rv = seadmapi_ObjGetNextInClass( &odf );
     }
  if ( ents > 0 )
     printf("Total of %d objects found in Class=%s\n", ents,
Class);
  else
     printf("Class %s, does not have any object.\n", Class);
  return 0;
}
 /* ------------------------------------------------------- */
 /* Display error message from security daemon.             */
 /* ------------------------------------------------------- */
static int ErrorMessage( int rv )
{ char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

More information:

seadmapi_ObjInClassList Function (see page 206)

seadmapi_ObjInClassList Function

The seadmapi_ObjInClassList function retrieves a list of objects in a specified class.

The field SEADMAPI_MAXOBJSLIST specifies the limit on the number of entries that can be retrieved in a single call.

After calling this function, you should call the seadmapi_FreeObjList function to free the memory allocated for the query. Use the ptr argument returned from this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_ObjInClassList (SEOSDB_CDF  *pcdf,
                    char        *start,
                    void        **ptr,
                    char        **names,
                    int         *count);
```

pcdf

A pointer to the class description.

start

A string representing the object name that should start the list.

ptr

A pointer to a "void *" that is used to free memory allocated for the list query.

names

A pointer to a vector of char pointers. Each element points to an object name.

count

On entry, the size of the names vector. On return, the number of entries in the vector.

More information:

seadmapi_FreeObjList Function (see page 207)
seadmapi_ObjGet Functions (see page 202)

seadmapi_FreeObjList Function

The seadmapi_FreeObjList function frees the memory allocated by the seadmapi_ObjInClassList function.

The function assigns NULL to *ptr.

There is no return value.

```
int seadmapi_FreeObjList (void **ptr);
```

ptr

The pointer as obtained by the most recent call to the seadmapi_ObjInClassList function.

More information:

seadmapi_ObjInClassList Function (see page 206)

seadmapi_FetchListPropVal Function

The seadmapi_FetchListPropVal function retrieves the values of a property that contains a list. The function cannot retrieve the value of a single-value property; use the seadmapi_FetchSinglePropVal function instead.

The function allocates a vector of void pointers, each pointing to an allocated buffer that holds a single element in the list. The caller has to define a variable of type void ** or any other type that is a pointer to a pointer (that is, int **). The caller sends a pointer to this variable, as shown in the following example.

```
{ int          **list;
  unsigned int   psize, count;
  int            rc;
  ...
rc = seadmapi_FetchListPropVal(..,
     (void ***)&list, &psize, &count);
```

Memory layout after call:

Argument	Description
[elem0]	1st data element
[elem1]	2nd data element
[elem*N*]	N+1 data element

where *elemN* is stored in the variable count.

All data fetched by this function is allocated and must be freed. To free the data, call the seadmapi_FreeListPropVal function.

The function can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_FetchListPropVal(const char    *szClass,
                     SEOSDB_CDF    *p_seclass,
                     const char    *szObj,
                     SEOSDB_ODF    *p_seobj,
                     const char    *szProp,
                     SEOSDB_PDF    *p_seprop,
                     void          ***val,
                     unsigned int  *psize,
                     unsigned int  *count);
```

szClass

The name of the class to which the object belongs. If the class is identified by the p_seclass parameter, set this parameter to NULL.

p_seclass

A pointer to a structure containing the class description. If the class is identified by the szClass parameter, set this parameter to NULL.

szObj

The name of the record whose property value is to be fetched. If the object is identified by the p_seobj parameter, set this parameter to NULL.

p_seobj

A pointer to the structure containing the object description. If the object is identified by the szObj parameter, set this parameter to NULL.

szProp

The name of the property whose value is to be fetched. If the property is identified by the p_seprop parameter, set this parameter to NULL.

p_seprop

A pointer to the structure containing the property description. If the property is identified by the szProp parameter, set this parameter to NULL.

val

A pointer to a pointer to a pointer of the type of value fetched. For more information, see the description following this list.

psize

The size of the fetched value.

count

The number of elements in the allocated vector.

Example

The following example demonstrates using the seadmapi_FetchListPropVal function to retrieve the values of a property that contains a list. This example also demonstrates the use of the seadmapi_gconn structure to display all the groups to which a user is linked.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                                 Version: 8.0
File          : sample_FetchList.c
Purpose          : Sample for seadmapi, Display a property that
            contains a list. Display list of groups user is
            connected to.
============================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
============================================================*/
#include <stdio.h>
#include <string.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])
{ SEOSDB_ODF        odf;
  SEOS_GCONN      **list;
  char              Object[ONAME_SIZE+1];
  unsigned int      elem_size;
  unsigned int      list_cnt;
  int               rv;
  int               cnt;
  if ( argc < 2 )
     { fprintf(stderr, "Required parameter (User Name)
missing.\n");
       return 1;
     }
  /* ------------------------------------------------------ */
  /* Set object name by specified parameter.                   */
  /* ------------------------------------------------------ */
  strcpy(Object, argv[1]);
  /* ------------------------------------------------------*/
  /* Get the list for class=USER, property=GROUPS,              */
  /*                  object=Specified_Parm                     */
  /* ------------------------------------------------------*/
  rv = seadmapi_FetchListPropVal("USER", NULL,
                                 Object,  NULL,
                                 "GROUPS", NULL,
                                 (void ***)&list,
                                 &elem_size, &list_cnt);
  /* ------------------------------------------------------ */
  /* Exit with error message in case we fail to get the list.  */
  /* ------------------------------------------------------ */
  if ( rv != 0 ) return ErrorMessage(rv);
  /* ------------------------------------------------------ */
  /* Display all groups from the list in a loop.               */
  /* ------------------------------------------------------ */
  for(cnt=0; cnt<list_cnt; cnt++)
```

```
        { rv = seadmapi_ObjGetEqual("GROUP", NULL, NULL,
                                   list[cnt]->oidGroup, &odf);
          if ( rv == 0 )
             { printf("Group Name: %-10s (id=%6ld)", odf.szOName,
                     list[cnt]->oidGroup);
             if ( list[cnt]->ugmUserMode
                { printf(", Group");
                if ( list[cnt]->ugmUserMode & SEOS_UGMODE_AUDITOR )
                     printf(" auditor");
                if ( list[cnt]->ugmUserMode & SEOS_UGMODE_PWMANAGER )
                     printf(" pwmanager");
                if ( list[cnt]->ugmUserMode & SEOS_UGMODE_ADMIN )
                   printf(" administrator");
                }
             else
                printf(", Regular");
             printf(".\n");
             }
        else
           printf("Group id: %ld, no longer exits in database.\n",
                 list[cnt]->oidGroup);
        }
  /* ------------------------------------------------------*/
  /* Free the list.                                             */
  /* ------------------------------------------------------ */
  seadmapi_FreeListPropVal((void ***)&list, &list_cnt);
  return 0;
}
/* ------------------------------------------------------ */
/* Display an error message from the security daemon.      */
/* ------------------------------------------------------ */
static int ErrorMessage( int rv )
{ char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

More information:

seadmapi_FetchSinglePropVal Function (see page 212)
seadmapi_FreeListPropVal Function (see page 219)

seadmapi_FetchSinglePropVal Function

The seadmapi_FetchSinglePropVal function retrieves the value of a property that contains a single value. The function cannot be used to retrieve lists; for properties that contain lists, use the function seadmapi_FetchListPropVal. To store the property's data, the calling program must allocate the space in memory pointed to by the val variable. To determine the size required, use the property descriptor sPVSize member or some other means.

The seadmapi_FetchSinglePropVal function can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are allowed to use this function.

This function can be used by any user who wants to view private data. The values are set to those of the user's own record.

This function can be used by any user to retrieve values for records of the SUDO class or the SEOS class.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_FetchSinglePropVal(const char  *szClass,
                     SEOSDB_CDF  *p_seclass,
                     const char  *szObj,
                     SEOSDB_ODF  *p_seobj,
                     const char  *szProp,
                     SEOSDB_PDF  *p_seprop,
                     void        *val,
                     int         *size);
```

szClass

The name of the class to which the object belongs. If the class is identified by the p_seclass parameter, set this parameter to NULL.

p_seclass

A pointer to a structure containing the class description. If the class is identified by the szClass parameter, set this parameter to NULL.

szObj

The name of the record whose property value is to be fetched. If the record is identified by the p_seobj parameter, set this parameter to NULL.

p_seobj

A pointer to the structure containing the object description. If the object is identified by the szObj parameter, set this variable to NULL.

szProp

The name of the property that is to be fetched. If the property is identified by the p_seprop parameter, set this variable to NULL.

p_seprop

A pointer to the structure containing the property description. If the property is identified by the sz_Prop parameter, set this variable to NULL.

val

A pointer to a location in memory where the result is to be stored.

size

On entry, this value is the size of the memory area pointed to by the parameter val. On return, this value is the size of the data stored in the memory area.

UNIX Example

In UNIX, the following example demonstrates the use of seadmapi_FetchSinglePropVal to retrieve the value of a single property.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 8.0
File          : sample_FetchSingle.c
Purpose          : Sample for seadmapi, List the value for a given
            property in a specific object in a specific class
            in a UNIX system.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])
{ SEOSDB_PDF    prop;                     /* Property Description */
  char          Class[CNAME_SIZE+1];
  char          Object[ONAME_SIZE+1];
  char          Property[PNAME_SIZE+1];
  char         *prop_val;
  int           data_size;
  int           rv;
  /* ----------------------------------------------------- */
  /* Check if SeOS_syscall is loaded.                     */
  /* ---------------------------------------------------- */
  rv = seadmapi_IsSeOSSyscallLoaded();
  if ( rv != 0 )
     { fprintf(stderr, "Database server is not running.\n");
       return 1;
     }

  /* ------------------------------------------------------ */
  /* Check if the user supplied all required parameters. */
  /* ---------------------------------------------------- */
  if ( argc < 4 )
     { fprintf(stderr, "Required parameter(s) missing.\n");
       fprintf(stderr, "Usage: '%s CLASS_NAME PROPERTY_NAME
                         OBJECT_NAME'\n", argv[0]);
       return 1;
     }
  /* ------------------------------------------------------ */
  /* Set the class, property, and object fields.            */
  /* ---------------------------------------------------- */
  strcpy(Class,    argv[1]);
  strcpy(Property, argv[2]);
  strcpy(Object,   argv[3]);
  /* ---------------------------------------------------- */
  /* Clear the property and object fields.                      */
  /* --------------------------------------------------- */
  memset( &prop, 0, sizeof(prop) );
  /* ----------------------------------------------------- */
  /* Get the property descriptor.                               */
  /* --------------------------------------------------- */
  rv = seadmapi_PropGetEqual( Class, NULL, Property, 0, &prop );
  if (rv) return ErrorMessage(rv);
     /* ----------------------------------------------------- */
     /* Check for string type.                                     */
     /* ----------------------------------------------------- */
  if ( prop.cPType != SEOSDB_PTYPE_STR )
     { fprintf(stderr, "This sample can display only character
                          values.\n");
       return 1;
     }
  /* ----------------------------------------------------- */
  /* Allocate memory for the value's data according to the */
  /*     property size                                      */
  /* --------------------------------------------------- */
  prop_val = (char *)malloc( (size_t)prop.sPVSize );
  if ( prop_val == NULL )
     { fprintf(stderr, "Failed to allocate required memory for
                          property value.\n");
       return 1;
     }
  data_size = prop.sPVSize;
  /* ----------------------------------------------------- */
  /* Get the requested property value.                          */
  /* --------------------------------------------------- */
  rv = seadmapi_FetchSinglePropVal(Class,  NULL,
                                   Object, NULL,
                                   NULL,  &prop,
                                   prop_val, &data_size);
  if (rv)
     { free(prop_val);
       return ErrorMessage(rv);
     }
  printf("%s\n", prop_val);
  free(prop_val);
  return 0;
}
/* ------------------------------------------------------ */
/* Display error message from security daemon. */
/* --------------------------------------------------- */
static int ErrorMessage( int rv )
```

```
{ char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

Windows Example

In Windows, the following example demonstrates the use of seadmapi_FetchSinglePropVal to retrieve the value of a single property.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 4.10
File          : sample_FetchSingle.c
Purpose          : Sample for seadmapi, List the value for a given
            property in a specific object in a specific class.
============================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
============================================================*/
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])
{ SEOSDB_PDF    prop;                    /* Property Description */
  char          Class[CNAME_SIZE+1];
  char          Object[ONAME_SIZE+1];
  char          Property[PNAME_SIZE+1];
  char         *prop_val;
  int           data_size;
  int           rv;
  /* ---------------------------------------------------- */
  /* Check if the user supplied all required parameters.  */
  /* ---------------------------------------------------- */
  if ( argc < 4 )
     { fprintf(stderr, "Required parameter(s) missing.\n");
       fprintf(stderr, "Usage: '%s CLASS_NAME PROPERTY_NAME
                           OBJECT_NAME'\n", argv[0]);
       return 1;
     }
  /* ---------------------------------------------------- */
  /* Set the class, property, and object fields.          */
  /* ---------------------------------------------------- */
  strcpy(Class,    argv[1]);
  strcpy(Property, argv[2]);
  strcpy(Object,   argv[3]);
  /* ---------------------------------------------------- */
  /* Clear the property and object fields.                */
  /* ---------------------------------------------------- */
  memset( &prop, 0, sizeof(prop) );
  /* ---------------------------------------------------- */
  /* Get the property descriptor.                         */
  /* ---------------------------------------------------- */
  rv = seadmapi_PropGetEqual( Class, NULL, Property, 0, &prop );
  if (rv) return ErrorMessage(rv);
     /* ------------------------------------------------ */
     /* Check for string type.                           */
     /* ------------------------------------------------ */
  if ( prop.cPType != SEOSDB_PTYPE_STR )
     { fprintf(stderr, "This sample can display only character
                        values.\n");
       return 1;
     }
  /* ---------------------------------------------------- */
  /* Allocate memory for the value's data according to the */
  /*     property size                                         */
  /* ----------------------------------------------------*/
  prop_val = (char *)malloc( (size_t)prop.sPVSize );
  if ( prop_val == NULL )
     { fprintf(stderr, "Failed to allocate required memory for
                        property value.\n");
       return 1;
     }
  data_size = prop.sPVSize;
  /* ---------------------------------------------------- */
  /* Get the requested property value.                          */
  /* -------------------------------------------------- */
  rv = seadmapi_FetchSinglePropVal(Class,  NULL,
                                   Object, NULL,
                                   NULL,  &prop,
                                   prop_val, &data_size);
  if (rv)
     { free(prop_val);
       return ErrorMessage(rv);
     }
  printf("%s\n", prop_val);
  free(prop_val);
  return 0;
}
/* ---------------------------------------------------- */
/* Display error message from security daemon.           */
/* ----------------------------------------------------*/
static int ErrorMessage( int rv )
{ char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

seadmapi_FreeListPropVal Function

The seadmapi_FreeListPropVal function must be used after fetching the list values using the seadmapi_FetchListPropVal function to free the memory allocated for the values. The parameters supplied to this function must be the same as those supplied to the seadmapi_FetchListPropVal function.

Any process can call this function.

There is no return value.

```
void seadmapi_FreeListPropVal(void ***list, unsigned int *count);
```

list

A pointer to the vector allocated by the seadmapi_FetchListPropVal function.

count

A pointer to the number of elements in the allocated vector.

More information:

seadmapi_FetchListPropVal Function (see page 207)

seadmapi_SetSinglePropVal Function

The seadmapi_SetSinglePropVal function sets the value of a single-value property. The function is used by the Watchdog and the agent. To prevent damage to the database, no other process is permitted to use this function.

The seadmapi_SetSinglePropVal function can be used only by the Watchdog and the agent.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_SetSinglePropVal(const char *szClass,
const char *szObj,
const char *szProp,
void       *val,
int        size);
```

szClass

The name of the class to which the record belongs.

szObj

The name of the record whose property is to be set.

szProp

The name of the property whose value is to be set.

val

The value to be assigned to the property.

size

The size, in bytes, of the value.

seadmapi_MakePDFList Function

The seadmapi_MakePDFList function retrieves the entire list of properties of a given class. The function allocates memory for the properties vector.

After using the seadmapi_MakePDFList function, free the allocated memory using the seadmapi_KillPDFList function.

These functions can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the seadmapi_MakePDFList function succeeds, it returns 0; if it fails, it returns an error code.

```
void seadmapi_KillPDFList(SEOSDB_PDF      **ppPdf,
   unsigned int    nCount);
int seadmapi_MakePDFList(const char      *szClass,
SEOSDB_PDF      **ppPdf,
unsigned int    *nCount);
```

szClass

The class name.

ppPdf

A pointer to the SEOSDB_PDF pointer that points to the allocated region of memory that holds the properties vector.

nCount

The number of properties in the vector.

seadmapi_Entity Functions

The seadmapi_GetEntity and seadmapi_GetExEntity functions retrieve into the ObjPVs vector all the values for the properties of a database object.

These functions are used by eTrust AC utilities and provide a convenient method to fetch the information from the database. For more information, see the rdbdump utility in the UNIX *Utilities Guide*.

To use these functions, first call the seadmapi_InitEntityRuler function to initialize the list of properties that are of interest to the caller. Next, call the seadmapi_GetEntity or seadmapi_GetExEntity function to fetch the information on a single object.

When you use the seadmapi_GetEntity function, the vector receives all the information about the property (the property description) and the property values. All properties are retrieved and stored as if they were list property values. Single-value properties are also stored as lists, with one entry.

When you use the seadmapi_GetExEntity function, the vector receives the same information, except that all property values that contain IDs of other objects are expanded. For example, instead of receiving an owner's ID, the function retrieves the expanded OID that contains the ID and the owner's class and name.

After using the information, call the seadmapi_KillEntityMem or seadmapi_KillExEntityMem function to free all memory required for the operation.

A user can initialize the ObjPVs vector from a previous call to the seadmapi_MakePDFList function. The vector pointed to by the ObjPVs parameter should contain the last element with the property name set to NULL. The functions use this method to determine the size of the vector.

The SEOSDB_ENTDAT, SEOS_X_OID, SEOS_X_GCONN, SEOS_X_ACL, and SEOS_X_PACL structures are defined in the seostypes.h header file.

To see the way data is fetched, see the notes for the seadmapi_FetchListPropVal function in this chapter.

After each call to seadmapi_GetEntity or seadmapi_GetExEntity, make sure you call seadmapi_KillEntityMem or seadmapi_KillExEntityMem, respectively, to free the memory allocated by the seadmapi_GetEntity or seadmapi_GetExEntity function.

These functions can be called by processes executed by users who have any of the following attributes:

- AUDITOR

- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_GetEntity(const char    *szCName,
                     const char    *szOName,
                     SEOSDB_ODF    *podf,
                     SEOSDB_ENTDAT *ObjPVs);
int seadmapi_GetExEntity(const char    *szCName,
                       const char    *szOName,
                       SEOSDB_ODF    *podf,
                       SEOSDB_ENTDAT *ObjPVs);

int seadmapi_InitEntityRuler(const char *szCName, SEOSDB_ENTDAT *ObjPvs);

void seadmapi_KillEntityMem(SEOSDB_ENTDAT *ObjPVs);

void seadmapi_KillExEntityMem(SEOSDB_ENTDAT *ObjPVs);
```

szCName

The class name.

szOName

The object name.

podf

A pointer to a memory area to be filled with the object description.

ObjPVs

A pointer to a vector with both property description and value list.

Example

The following example demonstrates the use of the seadmapi_InitEntityRuler and seadmapi_GetExEntity functions.

```
/*==========================================================
Project          : eTrust
Module           : eTrust                              Version: 8.0
File          : sample_TermOwn.c
Purpose          : Display terminal's owner.
==========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
==========================================================*/
/* --------------------------------------------------------
    When working in UNIX, remember to point an environment
    variable to the shared library path by entering the command
          setenv LD_LIBRARY_PATH/opt/CA/eTrustAccessControl/lib/
    ------------------------------------------------------- */
#include <ctype.h>
#include <stdio.h>
#include <sys/types.h>
#include <memory.h>
#include <seadmapi.h>
static int ErrorMessage( int rv );
int main(int argc, char *argv[])

{ SEOSDB_ODF          odf;                             /* Object definition */
  SEOSDB_ENTDAT       entdat[2];                       /* Entity data       */
  SEOS_X_OID          *owner;
  int                 rv;
/* ------------------------------------------------------- */
/* Check if user specified the terminal name.
*/
/* ------------------------------------------------------- */
  if ( argc == 1 )
     { printf("Usage: '%s terminal_name'\n", argv[0]);
       return 1;
     }
/* ------------------------------------------------------- */
/* Initialize entity data.
*/
/* ------------------------------------------------------- */
  memset(entdat, 0, sizeof(entdat));
/* ------------------------------------------------------- */
/* Set the ruler for the database request.
*/
/* ------------------------------------------------------- */
  entdat[0].szPName = "OWNER";              /* Owner             */
  entdat[1].szPName = NULL;                 /* Null terminator   */
  rv = seadmapi_InitEntityRuler("TERMINAL", entdat);
/* ------------------------------------------------------- */
/*  Exit with error message in case we fail to set the ruler.*/
/* ------------------------------------------------------- */
```

```
  if ( rv != 0 ) return ErrorMessage(rv);
/* ------------------------------------------------------- */
/* Get all data.
 */
/* ------------------------------------------------------- */
  rv = seadmapi_GetExEntity("TERMINAL",    /* Class name         */
                   argv[1],              /* Terminal name       */
                   &odf,                 /* Object definition   */
                   entdat);              /* Entity data         */
/* ------------------------------------------------------- */
/*  Exit with error message in case we fail to get the data */
/* ------------------------------------------------------- */
  if ( rv != 0 ) return ErrorMessage(rv);
/* ------------------------------------------------------ */
/*  Display OWNER information.
*/
/* ----------------------------------------------------- */
  if ( entdat[0].nPVQty != 0 )
     { owner = (SEOS_X_OID *)entdat[0].pPVList[0];
       if ( owner->pCName != NULL )
          printf("OWNER = %s %s, id=%d\n", owner->pCName,
                  owner->pOName, owner->oid);
       else
          printf("OWNER = (id=%d)\n", entdat[0].pPVList);
     }
  else
     printf("OWNER = \n");
  return 0;
}
/* ------------------------------------------------------- */
/* Display error message.                                */
/* ------------------------------------------------------- */
static int ErrorMessage( int rv )
{ char msg_buff[1024];
  seadmapi_GetMessage(rv, sizeof(msg_buff), msg_buff);
  fprintf(stderr, "%s.\n", msg_buff);
  return rv;
}
```

More information:

seadmapi_MakePDFList Function (see page 221)

seadmapi_GetGraceInfo Function

This function retrieves information regarding a user's password, date of last login, and the number of grace logins that the user still has.

The function can be called by processes executed by users who have the ADMIN attribute.

All users can execute this function for themselves.

```
int seadmapi_GetGraceInfo(SEGRACE_RES *p_sgr);
```

p_sgr

A pointer to the structure that contains the information regarding user logins and grace days.

UNIX Example

The following example demonstrates the use of the seadmapi_GetGraceInfo function in a UNIX system.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 4.10
File          : sample_grace.c
Purpose          : Sample for seadmapi: Display information from the
            Access Control database about the user's grace
            logins.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <sys/types.h>
#include <seadmapi.h>
int main(void)
{ SEGRACE_RES    sgr;  /*Grace information structure              */
  int            rv;   /*Return value                             */
  /*--------------------------------------------------------*/
  /*  Quit if the kernel extension is not running.               */
  /*--------------------------------------------------------*/
  rv = seossfr_IsSeOSSyscallLoaded();
  if ( rv )
     { fprintf(stderr, "The kernel extension is not loaded.\n");
       return rv;
     }
  /*--------------------------------------------------------*/
  /*  Quit if the security daemon is not running.               */
  /*--------------------------------------------------------*/
  rv = seadmapi_IsServerRunning();
  if ( rv )
     {fprintf(stderr, "Security daemon is not running.\n");
      return rv;
     }
  /*--------------------------------------------------------*/
  /*  Set username to '0' to get current user's             */
  /*  grace information                                     */
  /*--------------------------------------------------------*/
  sgr.uname[0] = 0;
  /*--------------------------------------------------------*/
  /*  Get grace information from the database.               */
  /*--------------------------------------------------------*/
  rv = seadmapi_GetGraceInfo(&sgr);
  /*--------------------------------------------------------*/
  /*  If rv is not zero, display an error message and quit. */
  /*--------------------------------------------------------*/
  if ( rv )
  { if ( sgr.step )
       { if (sgr.msg[0] != 0 )
```

```
           fprintf(stderr, "%s\n", sgr.msg);
          return 1;
      }
  }
 /*---------------------------------------------------------*/
 /*  Display the number of grace logins.                        */
 /*---------------------------------------------------------*/
 printf("User %s has %d grace logins.\n", sgr.uname, sgr.grace);
 return 0;
}
```

Windows Example

The following example demonstrates the use of the seadmapi_GetGraceInfo function in the Windows environment.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 4.10
File          : sample_grace.c
Purpose          : Sample for seadmapi: Display information from the
            Access Control database about the user's grace
            logins.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <sys/types.h>
#include <seadmapi.h>
int main(void)
{ SEGRACE_RES    sgr;  /*Grace information structure              */
  int            rv;   /*Return value                               */
  /*-------------------------------------------------------*/
  /*  Quit if the kernel extension is not running                 */
  /*-------------------------------------------------------*/
  rv = seossfr_IsSeOSSyscallLoaded();
  if ( rv )
     { fprintf(stderr, "The kernel extension is not loaded.\n");
       return rv;
     }
  /*-------------------------------------------------------*/
  /*  Set username to '0' to get current user's grace information*/
  /*-------------------------------------------------------*/
  sgr.uname[0] = 0;
  /*-------------------------------------------------------*/
  /*  Get grace information from the database.             */
  /*-------------------------------------------------------*/
  rv = seadmapi_GetGraceInfo(&sgr);
  /*-------------------------------------------------------*/
  /*  If rv is not zero, display an error message and quit. */
  /*-------------------------------------------------------*/
  if ( rv )
  { if ( sgr.step )
       { if (sgr.msg[0] != 0 )
            fprintf(stderr, "%s\n", sgr.msg);
           return 1;
       }
  }
 /*--------------------------------------------------------*/
 /*  Display the number of grace logins.                   */
 /*--------------------------------------------------------*/
 printf("User %s has %d grace logins.\n", sgr.uname, sgr.grace);
 return 0;
}
```

seadmapi_OidToName Function

This function provides a convenient way of translating an object ID to a string containing the object name. If the object does not exist in the database-for instance, if the object has been deleted-the string returned by this function is NULL.

The pointer returned by this function is a pointer to a static area, overwritten by each subsequent call. This makes the function unsafe when using multi-threads.

The function can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns the name of an object; if it fails, it returns NULL.

```
char *seadmapi_OidToName(SEOS_OID oid);
```

oid

The object ID of the record.

More information:

seadmapi_ObjGet Functions (see page 202)

seadmapi_WhoAmI Function

This function provides information about the current process to the Engine. The information returned by the function may be used to fetch other information from the database or from the Authorization Engine.

Things you should know about this function include:

- Each parameter is a pointer to a user variable that is assigned a value by the function. Any parameter may be NULL, in which case, it is not assigned a value.
- The handle returned by the function is an ACEE handle associated with the current process. Note that there may be more than one process with the same ACEE; in fact, this is usually the case, because each user has more than one process running in the system.
- The memory area pointed to by the szUName parameter must be large enough to accommodate 255 characters.
- The function is used by the sewhoami utility.
- The header file seadmapi.h contains several macros that operate on the objtype variable data to determine whether a user has a specific attribute. These macros have the command notation of SEOS_UMODE_is_attribute.
- Every process in the system can call this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_WhoAmI(uid_t *uid, int *handle, char *szUName, SEOS_UMODE *objtype);
```

uid

The UID associated with the current process. In UNIX, this is equivalent to and safer than using the getlogin UNIX function.

handle

The ACEE handle associated with the current process. See the Notes in this section.

szUName

The user name associated with the current process.

objtype

The user type as saved in the database. This specifies the attributes assigned to the user.

Example

The following example demonstrates the use of the seadmapi_WhoAmI function.

```
/*===========================================================
Project          : eTrust
Module           : eTrust                              Version: 8.0
Purpose          : Display information about the user from the
            Access Control database.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <ctype.h>
#include <stdio.h>
#include <sys/types.h>
#include <seadmapi.h>
int main(void)
{ SEOS_UMODE     objtype;        /* Object type     */
  uid_t
uID;            /* User ID         */
  char           uName[256];     /* User name       */
  int            handle;
  /* Handle          */
  int            regular = 1;    /* Mode flag       */
  int            rv;
/* -------------------------------------------------------- */
/*  Get user information from the database.
*/
/* -------------------------------------------------------- */
  rv = seadmapi_WhoAmI(&uID, &handle, uName, &objtype);
/* -------------------------------------------------------- */
/*  If failed, display message and quit.
*/
/* -------------------------------------------------------- */
  if ( (rv != 0) || (uName[0] == '?') )
     { fprintf(stderr, "Can't find current user name.\n",

uName, rv);
       return 1;
     }
/* -------------------------------------------------------- */
/*  Display the user information:
*/
/* -------------------------------------------------------- */
  printf("User Name   : %s\n",    uName);
  printf("User ID     : %ld\n",    uID);
  printf("User Handle : %ld\n", handle);
/* -------------------------------------------------------- */
/* Display user authorization attributes by using the
*/
/* SEOS_UMODE_is macro.
    */
/* ------------------------------------------------------- */
  printf("User Mode   :");
  if ( SEOS_UMODE_is_auditor(objtype) )
     { printf(" AUDITOR");
       regular = 0;
     }

  if ( SEOS_UMODE_is_operator(objtype) )
     { printf(" OPERATOR");
       regular = 0;
     }
  if ( SEOS_UMODE_is_admin(objtype) )
     { printf(" ADMIN");
       regular = 0;
     }
  if ( SEOS_UMODE_is_pwmanager(objtype) )
     { printf(" PWMANAGER");
       regular = 0;
     }
  if ( regular )
     printf(" REGULAR\n");
  else
     printf("\n");
  return 0;
}
```

seadmapi_WhoIs Function

The seadmapi_WhoIs function supplies information about the specified user. The function gets the user type-attribute-from the database. The memory area pointed to by the szUName parameter must be large enough to accommodate 255 characters.

The seadmapi.h header file contains several macros that operate on the objtype variable data to determine whether a user has a specific attribute. These macros have the common notation of SEOS_UMODE_is_attribute. Any process in the system can all this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_WhoIs(char *szUName, SEOS_UMODE *objtype);
```

szUName

The user name associated with the current process.

objtype

The user type as saved in the database. This specifies the attributes assigned to the user.

seadmapi_ACEE Function

When given a handle, seadmapi_GetACEE retrieves the relevant ACEE information. The function is also capable of scanning all ACEEs currently allocated for users in eTrust AC. The information is loaded into a memory area allocated by the function itself. The information filled in the CLIENT_ACEE structure contains all the credentials for a given ACEE.

Things you should know about this function include:

- Processes are scanned by the ACEE handle number. It is possible that scanning the ACEE handles will not return information on all allocated ACEEs because of logins or logouts that have occurred since the scan began.
- Every process in the system can use these functions. If a process wants to query information on a handle other than the current ACEE, it must have the ADMIN, SERVER, or AUDITOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_GetACEE(int hAcee, CLIENT_ACEE **ppAcee);
void seadmapi_FreeAceeMemory(CLIENT_ACEE **ppAcee);
```

ppAcee

A pointer to a pointer that is assigned to point to the memory area allocated by seadmapi_GetACEE. The function seadmapi_FreeAceeMemory receives this same address to free the allocated memory.

More information:

seadmapi_WhoAmI Function (see page 231)

seadmapi_GetMessage Function

This function retrieves an error description from the message file and places it in the buffer pointed to by the buff parameter.

Every process in the system can use this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_GetMessage(int err_code, int size, char *buff);
```

err_code

The error code as returned by one of the eTrust AC functions.

size

The size of the buffer in bytes. Normally a 2 KB buffer is sufficient.

buff

A pointer to a buffer that contains the text describing the error.

Example

For examples that use this function, see seadmapi_FetchListPropVal and seadmapi_FetchSinglePropVal in this chapter.

seadmapi_GetObjType Function

This function retrieves the object type stored in the current process's ACEE. This information can be used with several macros to determine whether the current process belongs to a user with one of the special attributes that can be assigned to a user-ADMIN, AUDITOR, PWMANAGER, and so forth.

The seadmapi.h header file contains several macros that operate on this variable data to determine whether a user has a specific attribute. These macros have the common notation of SEOS_UMODE_is_attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_GetObjType(SEOS_UMODE *objtype);
```

objtype

A pointer to a SEOS_UMODE type variable that is filled from the current process's ACEE.

Example

The following example shows how to use the seadmapi_GetObjType.

```
/*===========================================================
Project          : eTrust
Module          : eTrust                              Version: 8.0
File          : mymode.c
Purpose          : Sample for seadmapi_GetObjType
            Display user's mode:
            REGULAR AUDITOR ADMIN OPERATOR SERVER or PWMANAGER.
===========================================================
Copyright :
Copyright 2004 Computer Associates International, Inc.
===========================================================*/
#include <stdio.h>
#include <seostype.h>
#include <seadmapi.h>
int main(void)
{ int rv;
  SEOS_UMODE umode;
  if ( (rv = seadmapi_GetObjType(&umode)) == 0)
    {
      printf("My mode is 0x%x : ", umode);
      if ( umode != 0 )
        {
          if ( umode & SEOS_UMODE_AUDITOR )
            printf("Auditor ");
          if ( umode & SEOS_UMODE_OPERATOR )
            printf("Operator ");
          if ( umode & SEOS_UMODE_ADMIN )
            printf("Admin ");
          if ( umode & SEOS_UMODE_SERVER )
            printf("Server ");
          if ( umode & SEOS_UMODE_PWMANAGER )
            printf("PwManager ");
         }
      else
        printf("Regular ");
      printf("\n");
      return 0;
    }
  fprintf(stderr, "Error 0x%X for seadmapi_GetObjType.\n", rv);
  return 1;
}
```

More information:

seadmapi_WhoAmI Function (see page 231)

seadmapi_init Function

This function initializes the communication channel with eTrust AC.

Every process in the system can use this function. The function is not used in the Windows environment.

You must call the seadmapi_init or seadmapi_IsSeOSSyscallLoaded function before calling any other function in the seadmapi library. However, if you use both functions, seadmapi_init must precede seadmapi_IsSeOSSyscallLoaded.

This function returns 0 if initialization is successful or an error code on failure. It verifies the exact eTrust AC system call loaded on the computer.

```
int seadmapi_init(void)
```

seadmapi_IsSeOSSyscallLoaded Function

This function checks whether the eTrust AC system call is loaded. For Solaris, HP-UX, and Linux systems, it resolves the dynamic eTrust AC system call number.

This function can be used by every process in the system. It is not used in the Windows environment.

For all UNIX systems except AIX, if the function succeeds, it returns 0; if it fails, it returns an error code.

On AIX systems, the function always returns 0 because the AIX system loader cannot load any process that requires the eTrust AC system call to be loaded unless the system call actually *is* loaded.

```
int seadmapi_IsSeOSSyscallLoaded(void)
```

Example

For an example that uses this function, see seadmapi_FetchSinglePropVal in this chapter.

Note: You must call the seadmapi_init or seadmapi_IsSeOSSyscallLoaded function before calling any other function in the seadmapi library.

seadmapi_SendAuditRecord Function

The seadmapi_SendAuditRecord function sends any type of audit information to the audit log. This function is used internally by other functions provided with this API to submit specific types of audit log records to the log file. We recommend that you use the specific functions for each audit record type rather than using this function, although in several cases, the use of this one may be easier.

Things you should know about this function include:

- Whenever possible, use the specific audit function rather than the seadmapi_SendAuditRecord function.
- The function can be called by processes executed by users who have any of the following attributes:
  - ADMIN
  - AUDITOR
  - SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_SendAuditRecord(int type, int result, void *data);
```

type

The type of the audit record. For a list of valid values, see the selogtype.h header file.

result

One of the valid result codes supported by eTrust AC. For valid values, see the selogtype.h header file.

data

A pointer to the audit record data. This pointer must point to valid data according to the type of record being submitted.

seadmapi_SendAudit Functions

These functions send audit records to the audit log. The functions use the seadmapi_SendAuditRecord function (see page 239). We recommend that you use these functions rather than directly calling the seadmapi_SendAuditRecord function.

Following are the seadmapi_SendAudit functions:

int seadmapi_SendAdminAudit(SEOS_AUDITADMIN *rec, int result);

Sends audit records to the audit log in the format of administrative records.

int seadmapi_SendCwsAudit(SEOS_AUDITCWS *rec, int result);

Sends audit records to the audit log in the format of connect-with-service records.

int seadmapi_SendGenrAudit(SEOS_AUDITGENR *rec, int result);

Sends audit records to the audit log in the format of general resource records.

int seadmapi_SendInetAudit(SEOS_AUDITINWARN *rec, int result);

Sends audit records to the audit log in the format of TCP/IP records.

int seadmapi_SendLoginAudit(SEOS_AUDITLOGIN *rec, int result);

Sends audit records to the audit log in the format of login event records.

int seadmapi_SendShutdownAudit(SEOS_AUDITDOWN *rec, int result);

Sends audit records to the audit log in the format of shutdown records.

int seadmapi_SendStartupAudit(SEOS_AUDITSTART *rec, int result);

Sends audit records to the audit log in the format of startup records.

int seadmapi_SendUserAudit(SEOS_AUDITUSER *rec, int result);

Valid on UNIX only

Sends audit records to the audit log in the format of user records.

int seadmapi_SendWatchdogAudit(SEOS_AUDITWDWARN *rec, int result);

Sends audit records to the audit log in the format of Watchdog records.

eTrust AC uses a compression algorithm on the auditing information. Therefore, you should initialize the structure with 0s before filling in the information. The structure can be initialized by calling the memset function provided by the standard C library of every system.

The function can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns 0; if it fails, it returns an error code.

rec

A pointer to the structure containing event-specific data.

result

One of the valid result codes supported by eTrust AC. For a list of valid result codes, see the selogtype.h header file.

seadmapi_SendNotificationAudit Functions

These functions send notification records to the audit log. The functions use the seadmapi_SendAuditRecord function (see page 239). We recommend that you use these functions rather than directly calling the seadmapi_SendAuditRecord function. Notification records are generated for resources that have an associated user to notify for the event (NOTIFY property).

Following are the seadmapi_SendNotificationAudit functions:

int seadmapi_SendNfAdminAudit(SEOS_AUDITADMIN *rec, int result);

Sends notification records to the audit log in the format of administrative records.

int seadmapi_SendNfCwsAudit(SEOS_AUDITCWS *rec, int result);

Sends notification records to the audit log in the format of connect-with-service records.

int seadmapi_SendNfGenrAudit(SEOS_AUDITGENR *rec, int result);

Sends notification records to the audit log in the format of general resource records.

int seadmapi_SendNfInetAudit(SEOS_AUDITINWARN *rec, int result);

Sends notification records to the audit log in the format of TCP/IP records.

int seadmapi_SendNfLoginAudit(SEOS_AUDITLOGIN *rec, int result);

Sends notification records to the audit log in the format of login event records.

int seadmapi_SendNfShutdownAudit(SEOS_AUDITDOWN *rec, int result);

Sends notification records to the audit log in the format of shutdown records.

int seadmapi_SendNfStartupAudit(SEOS_AUDITSTART *rec, int result);

Sends notification records to the audit log in the format of startup records.

int seadmapi_SendNfUserAudit(SEOS_AUDITUSER *rec, int result);

Valid on UNIX only

Sends notification records to the audit log in the format of user records.

int seadmapi_SendNfWatchdogAudit(SEOS_AUDITWDWARN *rec, int result);

Sends notification records to the audit log in the format of Watchdog records.

eTrust AC uses a compression algorithm on the auditing information. Therefore, you should initialize the structure with 0s before filling in the information. The structure can be initialized by calling the memset function provided by the standard C library of every system.

The function can be called by processes executed by users who have any of the following attributes:

- AUDITOR
- SERVER

The Watchdog and the agent are also allowed to use these functions.

If the function succeeds, it returns 0; if it fails, it returns an error code.

rec

A pointer to the structure containing event-specific data.

result

One of the valid result codes supported by eTrust AC. For a list of valid result codes, see the selogtype.h header file.

seadmapi_SendErrorLog Function

This function is used by the Watchdog and the agent to place a trace-back to note a possible error or malfunction. The error description is common for all error records in the error log file.

- eTrust AC uses a compression algorithm on the error information. Therefore, you should initialize the structure with '0's before filling in the information. The structure can be initialized by calling the memset function provided by the standard C library of every system.
- The seadmapi_SendErrorLog function can be used only by the Watchdog and the agent.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_SendErrorLog(SEOS_REQ_ERRORDESCP *rec);
```

rec

A pointer to the structure containing a description of the error and the trace-back data.

seadmapi_ProcessControl Function

This function is used by a process to control its auditing level and security. Any process can call this function to turn on auditing for all operations or to delete the credentials associated with the process. Using the flags parameter, a process can control these values. The flags parameter can contain any of the values described in the following table or a bit-wise OR value of them.

The following flags are currently supported:

SEADMAPI_PROCCNTL_NOACEE

A request by a process to remove its ACEE and use the credentials of a user who is not defined to eTrust AC. The new NULL credentials are assigned to the process and any child processes.

SEADMAPI_PROCCNTL_LOGALL

A request to audit every request made by the process and its child processes.

The senone utility uses this function; every process in the system can call it.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_ProcessControl(unsigned long flags);
```

flags

One or more bit-wise values.

seadmapi_consTrace Functions

These functions control the trace logging. The trace function is used to help diagnose problems and to help understand how eTrust AC behaves.

All functions return the trace status after the call. A value of 1 means the trace is enabled, while 0 means the trace is disabled. If the CurrStatus parameter specified to any of these functions is NULL, the function does not fill in current status of the trace.

These functions and the -t option of the secons utility provide the same functionality.

Things you should know about this function include:

- Trace status is not maintained across seosd sessions.
- These functions can be called by users who have the ADMIN or OPERATOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consTraceClear(int *CurrStatus);
int seadmapi_consTraceDisable(int *CurrStatus);
int seadmapi_consTraceEnable(int *CurrStatus);
int seadmapi_consTraceGetStatus(int *CurrStatus);
int seadmapi_consTraceToggle(int *CurrStatus);
```

CurrStatus

Status of the trace after the call.

seadmapi_consUidLogin Functions

These functions operate on a user's concurrent login setting. The functions do the following:

- Disable concurrent logins for the user
- Enable concurrent logins for the user
- Retrieve the current concurrent logins setting for the user

The user is identified by the uid parameter.

These functions and the -d and -u options of the secons utility provide the same functionality.

The seadmapi_consUidLoginDisable function disables concurrent logins for the specified user ID. The seadmapi_consUidLoginEnable function reenables concurrent logins for the specified user ID.

The seadmapi_consUidLoginGetStatus function retrieves the status of the user's concurrent logins setting as provided by the authorization daemon.

Things you should know about this function include:

- Enabling or disabling concurrent logins is not maintained across seosd sessions.
- eTrust AC provides an enforced concurrent logins mechanism that may be more suitable for most sites. Use the maxlogins parameter of the setoptions command to set the maximum number of currently logged-in terminals for every user on the system. (If you give maxlogins a zero value, setoptions does not enforce a maximum.)
- Users can disable or enable concurrent logins for themselves.
- Users with the ADMIN attribute can disable concurrent logins for any user.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consUidLoginDisable(int uid);
int seadmapi_consUidLoginEnable(int uid);
int seadmapi_consUidLoginGetStatus(int uid, nt *CurrStatus);
```

uid

The user ID on which the function is to operate.

CurrStatus

The current status of the user's concurrent logins setting.

seadmapi_consAllLogin Functions

These functions control users' ability to log into the system. If login is disabled, no user is permitted to log into the system while eTrust AC is running.

These functions and secons -L option provide the same functionality.

Things you should know about this function include:

- The root user is always allowed to log in and is not subject to disabling by this global flag.
- Enabling or disabling login ability is not maintained across seosd sessions.
- These functions can be called only by users who have the ADMIN or OPERATOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consAllLoginDisable(void);
int seadmapi_consAllLoginEnable(void);
int seadmapi_consAllLoginGetStatus(int *CurrStatus);
```

CurrStatus

The current status of the system-wide disable login flag.

seadmapi_consRefreshIPAddresses

This function refreshes the host name to IP address resolution at the kernel run time tables.

If the function succeeds, it returns 0; if it fails, it returns an error code.

Note: This function can only be called by users who have the ADMIN attribute.

```
int seadmapi_ReloadIni(unsigned int* puiNumberOfRefreshedResources);
```

puiNumberOfRefreshedResources

When the function completes successfully, holds the number of resources that have been refreshed.

seadmapi_consRunTimeStatisticsGet Function

This function retrieves runtime statistics on the seosd. This information can be viewed using the secons utility. The information is placed in the structure pointed to by the rtsStat parameter. The structure contains the following information:

inet_deny

The number of TCP/IP requests that were denied.

inet_grant

The number of TCP/IP requests that were granted.

inet_error

The number of TCP/IP requests that could not be resolved because of an error.

audit_log_q

The number of unwritten audit records in the queue.

error_log_q

The number of unwritten error records in the queue.

oidLast

Last used object ID.

pidLast

Last used property ID.

cidLast

Last used class ID.

classRecCount

Number of classes in the database.

propRecCount

Number of properties in the database.

objRecCount

Number of objects (records) in the database.

pvRecCount

Number of records in the properties-values database.

nAceeHandles

Number of ACEE entries currently used.

nClients

Number of protected clients. (This field is reserved for future use.)

nTrusted

Number of trusted programs loaded into the cache.

nUntrusted

Number of programs marked as non-trusted in the cache.

This function and the -i option of the secons utility provide the same information.

Things you should know about this function include:

- The information retrieved by this function may be extended in future versions of eTrust AC; however, backward compatibility will always be maintained. The client program, therefore, gets only that information that is required. The size of the structure is an indication of the version of eTrust AC in use. Any additional information is added at the end of this buffer.
- This function can only be called by users who have the ADMIN or OPERATOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consRunTimeStatisticsGet SEADMAPI_RTSTAT *rtsStat);
```

rtsStat

A structure containing the run-time statistics, as discussed in the description.

seadmapi_consMessageSend Function

This function submits a message to the eTrust AC trace.

This function and the -m option of the secons utility provide the same functionality.

Any process can call this function.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consMessageSend(const char *szMessage);
```

szMessage

String of the message to place in the trace log.

seadmapi_consShutdown Function

This function causes eTrust AC to exit and disables the three daemons that compose the eTrust AC basic configuration-seosd, seoswd, and seagent. (seagent is also known as the agent.) After shutting down the eTrust AC UNIX daemons (Windows services), the kernel extension remains loaded but is not active until seosd is executed again. This disables all protection provided by eTrust AC.

In UNIX, other daemons that are part of eTrust AC, such as serevu, selogrd, and selogrcd are not affected by this function. These processes can be killed explicitly.

The seadmapi_consShutdown function and the -s option of the secons utility provide the same functionality.

This function can only be called by users who have the ADMIN or OPERATOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_consShutdown(void);
```

seadmapi_ReloadIni Function

This function reloads the configuration tokens of the seosd daemon. The tokens are in the seos.ini file. The daemon actually uses only part of the newly reloaded tokens.

This function can only be called by users who have the ADMIN or OPERATOR attribute.

If the function succeeds, it returns 0; if it fails, it returns an error code.

```
int seadmapi_ReloadIni(void);
```

sepass_ReplacePassword Function

The sepass_ReplacePassword function replaces the user password with a new password.

Assuming the user is defined locally, and SEPASS_API_DOMAIN_PMD is not given in the domainCode arguments, sepass_ReplacePassword replaces the user's password in the local UNIX file.

The password can be replaced according to these criteria:

- In the local security database, assuming that SEPASS_API_DOMAIN_PMD is not given in the domainCode arguments
- If the Policy Model confirms the password file
- If a Policy Model is given
- If a Policy Model is defined in the seos.ini file
- If a Policy Model is specified for the user or its profile group and the SEPASS_API_DOMAIN_LOCAL is not specified
- If the PMDB is given
- If the PMDB is defined in the seos.ini file or specified for the user or its profile group, and the SEPASS_API_DOMAIN_LOCAL is not specified
- If the NIS+ table and the nis_env token is set to niplus

```
int sepass_ReplacePassword (char *userName,
                    char *oldPasswd,
                    char *newPasswd,
                    char *szPmd,
                    char* szMsg,
                    int msgLen,
                    int domainCodes,
                    int debug,
                    int ignoreRules,
                    int keep_grace,
                    int do_as_user,
                    int onlySeOS,
                    int ChangeGrace);
```

userName

A NULL-terminated string containing the name of the user whose password is to be replaced.

oldPasswd

A NULL-terminated string containing the current (old) password of the named user or the administrator invoking the function.

newPasswd

A NULL-terminated string containing the new (desired) password.

szPmd

A NULL-terminated string containing the name of the Policy Model on which to change the password model (if any).

szMsg

A pointer to the buffer in which the success for failure messages will be stored.

msgLen

The size of the message buffer.

domainCodes

One of these values:

- #define SEPASS_API_DOMAIN_LOCAL
- #define SEPASS_API_DOMAIN_PMD
- #define SEPASS_API_DOMAIN_ALL
- debug

A flag indicating the detailed description that should be printed.

ignoreRules

This argument can have one of three values:

- **IGNORE_NEVER**-never ignore password policy rules.
- **IGNORE_ALWAYS**-always ignore password policy rules.
- **IGNORE_ADMIN**-ignore password policy rules only in administrative change.

keep_grace

Reset the grace attribute for the user after passwords.

do_as_user

Change the password as if the named user is changing it, and not as an administrative change.

onlySeOS

This argument can have one of the following two values:

1 - Change password only in eTrust AC environment;

0 - Change password in UNIX and eTrust AC environments.

ChangeGrace

Set the users' grace period for login to the specified number of days.

-1 => use the default (1).

Structures and Data Types: eTrust AC Database Data Structures

In the database, each property has its data type. Some data types defined in the database are simple, such as strings or integer values; others are structures with several fields.

The following section describes the major data structures and data types.

Each of the data types eTrust AC defines and saves in the database has a symbolic constant defined using the following naming convention:

SEOSDB_PTYPE_Access_Control_data_type

For example, the type SEOS_OID has a symbolic constant of SEOSDB_PTYPE_OID.

C Data type or PTYPE

SEOSDB_PTYPE_STR

An ASCII-Z string. Size of property determined by the maximum length of the string.

SEOSDB_PTYPE_OID

An object ID in the database.

SEOSDB_PTYPE_TIME

A date and time. (The format is time_t.)

SEOSDB_PTYPE_INT

An integer, with 1, 2, or 4 bytes. The property size determines the actual type.

SEOSDB_PTYPE_UINT

An unsigned integer, with 1, 2, or 4 bytes. The property size determines the actual type.

SEOSDB_PTYPE_UMODE

User mode, that is, whether a user is ADMIN, AUDITOR, and so on. The value is stored in an integer as a bit field.

SEOSDB_PTYPE_GCONN

The connection information of a user to group.

SEOSDB_PTYPE_ACCS

The access type allowed. The access is a 32-bit field value.

SEOSDB_PTYPE_ACL

An ACL entry.

SEOSDB_PTYPE_PACL

A program conditional ACL entry.

SEOSDB_PTYPE_UAUDITM

The audit triggers for a user account. It is a bit field value.

SEOSDB_PTYPE_DAYTIME

A day and time restriction on access to a resource.

SEOSDB_PTYPE_TRPINFO

A trusted program information of the file characteristics.

SEOSDB_PTYPE_INETACL

An ACL entry for HOST and other internet classes.

SEOSDB_PTYPE_RAUDITM

The audit triggers for a resource. It is a 16-bit field value.

SEOSDB_PTYPE_BOOL

An unsigned char that represents a Boolean value.

SEOSDB_PTYPE_IPMSKMTCH

A combination of a mask and match values for IP addresses.

SEOSDB_PTYPE_INSRACL

A TCP/IP services range ACL entry.

Description

The information stored in each and every data member of these structures and the values are listed in the *seostypes.h* file. In some of the cases, the structure is only a single value or a list of entries each of the same type. For example, an ACL is a list of 0 or more ACL entries. The list of groups a user is connected to is a list of SEOS_GCONN structures. The latter case also represents a case in which entities are pointing one to another. The user object has the list of groups to which it is connected. The group has the list of users connected to it.

For a complete description of each property in every class, see the *Reference Guide*.

The following database structures and data types are described in this and other chapters as indicated:

CLIENT_ACEE

Contains the information for a given ACEE^1

SEADMAPI_RTSTAT

Contains the runtime statistics^1

SEGRACE_RES

Contains the grace login information^1

SEOS_ACCS

Contains a list of access flags^2

SEOS_ACL

Contains a list of ACLs^1

SEOS_CID

Contains the class identification descriptor^3

SEOS_GCONN

Contains a list of groups to which a user is connected and the attributes the user has, if any, with respect to each group^1

SEOS_OID

Contains the object identification descriptor^3

SEOS_PID

Contains the property identification descriptor^3

SEOS_X_ACL

Contains a list of ACLs with more information that SEOS_ACL^1

SEOS_X_GCONN

Similar to SEOS_GCONN but contains more information[1]

SEOS_X_OID

Holds an expanded object identification descriptor[1]

SEOS_X_PACL

Contains an expanded list of PACLs[1]

SEOSDB_CDF

Contains the definition of a specific class in the database[1]

SEOSDB_ENTDAT

Contains information about a property in the database[1]

SEOSDB_ODF

Contains the definition of a specific object in the database[3]

SEOSDB_PDF

Contains the definition of a specific property in the database[3]

[1] Described in this chapter

[2] Described in the chapter ”Authorization and Authentication API”

[3] Described in the chapter “Exits API”

Log File Structures

eTrust AC uses a binary compressed file to store audit and error log records. The log files consist of a fixed portion and a variable-length portion for every record. The fixed portion contains some commonly used data (like time of record submission) and the size and type of the variable-length portion. eTrust AC defines several structures for different auditing records. The current version of eTrust AC provides the following audit records.

Data Structures

The following structures are described in the chapter "Language Client API."

SEOS_AUDITADMIN

eTrust AC updates and language parser interface requests

SEOS_AUDITDOWN

eTrust AC daemon and service shutdown events

SEOS_AUDITGENR

General resource audit events

SEOS_AUDITINWARN

eTrust AC TCP/IP events auditing information

SEOS_AUDITLOGIN

Login events and password events audit record

SEOS_AUDITSTART

eTrust AC daemon and service startup events

SEOS_AUDITUSER

Trace records for audited users

SEOS_AUDITWDWARN

Watchdog audit records for marking trusted programs as untrusted

SEOS_REQ_ERRORDESCP

eTrust AC error trace-back information

All these structures are defined in the header file selogtype.h.

CLIENT_ACEE Structure

The CLIENT_ACEE structure contains information for a given ACEE. The fields are:

long hAcee

The handle of the ACEE.

long nGroups

The number of group connections.

CLIENT_ACEE_GCONN *Groups

The group connection array.

long nCategories

The number of categories.

char *pszCategories

The array of category names.

char *szSecLabel

The security label.

char *szUsername

The user's name.

SEOS_UAUDIT_MODE AuditMode

The user's audit mode.

unsigned char SecLevel

The user's security level.

char *szTerminal

The source terminal.

int count

The process count for the ACEE.

SEOS_UMODE user_mode

The user's mode.

time_t create_time

The creation time for the ACEE.

SEADMAPI_RTSTAT Structure

The SEADMAPI_RTSTAT structure contains information retrieved from the seadmapi_consRunTimeStatisticsGet function.

The fields are:

unsigned int inet_deny

The number of inet requests that were denied.

unsigned int inet_grant

The number of inet requests that were granted.

unsigned int inet_errors

The number of inet requests that contained errors.

long audit_log_q

The audit log queue size.

long error_log_q

The error log queue size.

SEOS_OID oidLast

The first free object ID in the database.

SEOS_PID pidLast

The first free property ID in the database.

SEOS_CID cidLast

The first free class ID in the database.

long classRecCount

The number of classes in the database.

long propRecCount

The number of properties in the database.

long objRecCount

The number of objects in the database.

long pvRecCount

The number of property values in the database.

long nAceeHandles

The number of ACEE handles currently in the system.

long nClients

The number of protected clients.

long nTrusted

The number of trusted programs currently in the database.

long nUnTrusted

The number of untrusted programs currently in the database.

SEGRACE_RES Structure

The SEGRACE_RES structure contains information retrieved by the seadmapi_GetGraceInfo function.

The fields are:

int step

An integer that represents the type of information contained in the structure. Values include:

SEGRACE_STEP_NONE

No data is available to display.

SEGRACE_STEP_WARN

Display a warning to the user such as the number of grace days the user has left or the number of days until the user's password expires.

SEGRACE_STEP_MUST

The user's password has expired; it must be replaced now.

char msg[ ]

The message that appears for the user.

char last_log[ ]

A message containing information regarding the user's last login.

int grace

The number of grace logins left to the user.

int days

The number of days until the user must replace the password.

char uname[ ]

The name of the user for whom the inquiry is done.

SEOS_ACL Structure

The SEOS_ACL structure describes the access permitted to the accessor.

The fields are:

SEOS_OID oidAccessor

The object ID of the accessor.

SEOS_ACCS Accs

The user's level of access to the resource. For a list of available access types, see the SEOS_ACCS data type in the chapter "Authorization and Authentication API."

SEOS_GCONN Structure

The SEOS_GCONN structure contains the list of groups to which the user is connected and the special attributes, if any, that the user has with respect to those groups.

The fields are:

SEOS_OID oidGroup

The object ID of the group.

SEOS_OID oidAuthor

The owner of the connection.

SEOS_TIME tConn

The date and time the connection was established.

SEOS_UGMODE ugmUserMode

The attributes of the user in the group.

SEOS_PACL Structure

The SEOS_PACL structure contains a list of conditional access control lists.

The fields are:

SEOS_OID oidAccessor

The object ID of the accessor.

SEOS_OID oidProg

The object ID of the trusted program.

SEOS_ACCS Accs

The user's level of access to the resource. For a list of available access types, see the SEOS_ACCS data type in the chapter “Authorization and Authentication API.”

SEOS_REQ_ERRORDESCP Structure

The SEOS_REQ_ERRORDESCP structure contains information sent to the audit log.

The fields are:

int module

The number of the module that sent the error record.

int code

The error code.

char name[ ]

The name of the module that sent the error record.

char source[ ]

The source file of the error.

int stage

The stage in which the error was noticed.

int severity

The severity of the error.

SEOS_X_ACL Structure

The SEOS_X_ACL structure is an expanded version of the SEOS_ACL structure.

The fields are:

SEOS_OID oidAccessor

The object ID of the accessor.

SEOS_ACCS Accs

The user's level of access to the resource. For a list of available access types, see the SEOS_ACCS data type in the chapter “Authorization and Authentication API.”

char *pAccessorCName

The class to which the accessor belongs.

char *pAccessorOName

The accessor's name.

SEOS_X_GCONN Structure

The SEOS_X_GCONN structure is an expanded version of the SEOS_GCONN structure.

The fields are:

SEOS_OID oidGroup

The object ID of the group.

SEOS_OID oidAuthor

The owner of the connection.

SEOS_TIME tConn

The date and time the connection was established.

SEOS_UGMODE ugmUserMode

The attributes of the user in the group.

char *pGName

The group's name.

char *pAuName

The author's name.

SEOS_X_PACL Structure

The SEOS_X_PACL structure is an expanded version of the SEOS_PACL structure.

The fields are:

SEOS_OID oidAccessor

The object ID of the accessor.

SEOS_OID oidProg

The object ID of the trusted program.

SEOS_ACCS Accs

The user's level of access to the resource. For a list of available access types, see the SEOS_ACCS data type in the chapter “Authorization and Authentication API.”

char *pAccessorCName

The class name.

char *pAccessorOName

The object name.

char *pProgName

The name of the program through which access is allowed.

SEOSDB_CDF Structure

The SEOSDB_CDF structure contains the definition of a specific class in the database.

The fields are:

SEOS_CID sCId

The ID of the class.

char szCName[]

The name of the class.

unsigned long lCFlags

The flags of the class.

unsigned char cCRLevel

Reserved for future use.

unsigned char cCWLevel

Reserved for future use.

char reserved[]

Reserved for future use.

SEOSDB_ENTDAT Structure

The SEOSDB_ENTDAT structure contains the information returned by the seadmapi_GetEntity and seadmapi_GetExEntity functions.

The fields are:

char *szPName

The property name.

SEOS_CID sCId

The class ID of the object's class.

SEOS_PID sPId

The property ID of the object.

unsigned long int lPFlags

The property's flags.

unsigned short int sPVSize

The property's size.

unsigned char int cPType

Reserved for future use.

unsigned char cPRLevel

Reserved for future use.

unsigned char cPWLevel

Reserved for future use.

unsigned int nPVQty

The number of values in pPVList.

void **pPVList

The value list.

unsigned int nErrorCode

The error code if the function was not able to return the requested data.

SEOS_X_OID Structure

The SEOS_X_OID data type is an expanded version of the SEOS_OID data type.

The fields are:

SEOS_OID oid

The object ID of the record as an unsigned long integer.

char *pCName

The object's class.

char *pOName

The object's name.

Chapter 7: IR API

This section contains the following topics:

The IR API

This library supplies an interface to eTrust AC log files - seos.audit and seos.audit.bak. You can set whether the IR API library routes audit events of existing PMDs in addition to the local security events by setting the irecorder_audit token in the Windows registry or the UNIX seos.ini file.

This API enables an external application to view audit records in chronological order.

Note: The library only supports PMD audit events that are created on the same computer where eTrust AC is installed.

Important! The IR API is not safe for multi-threading.

Structures

Data Position

A void handle that an API function operates according to its value.

```
typedef struct
{
        int ir_ver;             eTrust AC IRecorder version
        int ent_cnt;            Number of audit hosts
        int curr_idx;           Current audit host
        void *ents;             Pointer to host entries
} IRApiDataPosition;            Interpreted audit record
```

UTF8 Interpreted Audit Record

Contains two string fields that describe an eTrust AC audit record field. A null terminated list of SEOS_UTF8AUDLOGINTERP structures represents one eTrust AC audit event record.

```
typedef struct tagSeosUTF8LogInterpreted
{
        char *Label;            Label of an audit field in the record
        char *Value;            Value assigned in the audit record to the field
} SEOS_UTF8AUDLOGINTERP;        Interpreted audit record
```

Functions

int eacIRApi_LogInit(IRApiDataPosition *pos);

This function:

1. Gets the audit file names (from the eTrust AC or PMD settings).
2. Resets the offset in order to get the oldest log file, and opens it to read the audit record.
3. Initializes the IRApiDataPosition pointer for calling eacIRApi_LogGetNext().
4. Initializes threads and events for IR API and PMD synchronization.

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_PART_FAIL	Partial failure (an audit file could not be initialized)
IR_EAC_GENERAL_FAIL	General failure - IR API should be initialized

***pos**

(OUT) A pointer to IRApiDataPosition structure.

int eacIRApi_LogGetNext(IRApiDataPosition *pos, SEOS_UTF8AUDLOGINTERP **ppUtfMsg, int *log_type);

Returns the next log record and its type according to a given data position.

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_NO_MORE_DATA	No more data in all audit files (PMD and localhost)
IR_EAC_DATA_LOST	Data lost
IR_EAC_PSBLY_DATA_LOST	Possibly data lost
IR_EAC_GENERAL_FAIL	Failure in current audit file

***pos**

(IN) A pointer to IRApiDataPosition - represents the position to get the next log record from (eTrust AC or PMD audit file and offset).

****ppUtfMsg**

(OUT) A pointer to (UTF8 strings) interpreted record.

***log_type**

(OUT) A pointer to an integer representing the next log record type.

int eacIRApi_LogTerminate(void);

Closes all log files for read, terminates all threads and reset events.

Call this function when you are finished working with the API.

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_PART_FAIL	Partial failure (some audit files could not be closed)
IR_EAC_GENERAL_FAIL	General failure

int eacIRApi_LogReset(void *buff, int size);

Opens log file (if it still exists) according to a given data position. It enables you to go back to an old (saved) state and continue from there.

Note: To create the data for using parameter 1 (void *buff), use the function eacIRApi_CopyDataPosition (see page 273).

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_PART_FAIL	One or more audit files could not be initialized
IR_EAC_GENERAL_FAIL	Library should be terminated

***buff**

(IN) A pointer to the buffer that contains the data needed for resetting the new data position.

size

(IN) An integer representing the data size.

void eacIRApi_LogFreeInterpretRecord(void);

This function frees the last interpreted record.

There are no return values for this function.

Note: eacIRApi_LogGetNext (see page 271) calls the same internal function each time it runs, so you do not need to use eacIRApi_LogFreeInterpretRecord before calling for next log record.

char *eacIR_LogGetVersion(void);

Returns the API version (according to the eTrust AC build number) as a string.

The return value is the version number as a string.

int eacIRApi_CopyDataPosition(void **pos)

Copies the current audit position to the allocated memory position specified. You need to use eacIRApi_GetDataPositionSize (see page 273) to return the size of the memory that should be allocated to the position specified.

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_GENERAL_FAIL	Library should be terminated

****pos**

(OUT) The address of the buffer pointer where the copied data is located.

size_t eacIRApi_GetDataPositionSize(void)

Returns the size of the allocated memory needed for copying the current data position.

int eacIRApi_GetLastError(SEOS_UTF8AUDLOGINTERP **pUtfMsg)

Returns the last error description. You should call this function whenever another function returns IR_EAC_GENERAL_FAIL. A description of the last error is allocated in an array of SEOS_UTF8AUDLOGINTERP structures (in the same way the audit record is allocated for the eacIRApi_LogGetNext function).

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_GENERAL_FAIL	Library should be terminated

****pUtfMsg**

(OUT) A pointer to a UTF8 strings last error description.

int eacIRApi_ConvertOldData(void *oldbuff, size_t oldsize, void **newbuff, size_t *newsize)

Converts old data position, saved with the obsolete IR API (see page 281), so that it is suitable for the new IR API (when upgrading IRecorder).

Possible return values are:

Value	Meaning
IR_EAC_SUCCESS	Success
IR_EAC_GENERAL_FAIL	Library should be terminated

***oldbuff**

(IN) A pointer to the old buffer.

oldsize

(IN) The size of the old buffer.

****newbuff**

(OUT) A pointer to the new buffer.

***newsize**

(OUT) The size of the new buffer.

Appendix A: tcllca: The LCA Extension

This section contains the following topics:

The tcllca.so Library

This appendix describes the tcllca.so library, an LCA extension that adds LCA commands to the TCL environment.

Note: This information is valid for UNIX only.

Programming Guide

More flexible than the selang command language, tcllca imitates selang from the TCL shell environment while adding new commands to return information about eTrust AC objects-users, resources, classes, and properties-from the TCL environment, without using a selang command and parsing the result.

The extension is in shared library format, loaded by the TCL load command. You may need to set the LD_LIBRARY_PATH to the lib subdirectory in the eTrust AC directory. To successfully load the extension, you must have the eTrust AC admin flag and access to the local terminal, and be running eTrust AC.

The extension loads the following shared libraries:

- liblangapi
- libseadmapi
- libselang

To load the tcllca extension:

1. If necessary, set the library path with the following command where *eTrustACDir* is the directory where eTrust AC is installed:

   ```
   setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:eTrustACDir/lib
   ```

2. Load the TCL shell with the command:

   ```
   tclsh
   ```

   The % prompt appears.

   Load the tcllca library with the command:

   ```
   load eTrustACDir/tcllca.so
   ```

Sample Program

The following sample program first loads the tcllca.so extension library, and then finds all the users owned by the root user.

```
>setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/opt/CA/eTrustAccessControl/lib
>tclsh
%load /opt/CA/eTrustAccessControl/lib/tcllca.so
%set user_list [se_get_resources user]
%foreach user $user_list {
  if {[se_scan_props user $user OWNER] == “OWNER root”} {
     lappend root_owned $user
  }
 }
%if [info exists root_owned]  {
   puts “users owned by root : $root_owned”
 } else {
   puts “no users owned by root”
 }
%
```

tcllca Functions

The LCA TCL extension includes the following functions:

se_class_list Function

The se_class_list function prints all the classes in the database.

```
se_class_list
```

se_get_resources Function

The se_get_resources function prints the names of all the objects in the specified class. If you specify an object, the function searches for it in the database and, if the object does not exist, returns an error. If the object does exist, the function prints the name of the object. If the object contains wildcards, the function returns the object list.

For example, the following command returns a list of all the users in the database:

```
se_get_resources user
se_get_resources class [object];
```

segetstat Function

The segetstat function tells you the status returned by the last command:

OK

The last command was successfully processed.

ERROR

The last command was not successfully processed.

DUP

Relevant only for the newfile, newgrp, newres, and newusr commands. The object already exists inside the database.

NOTICE

The information message was returned from a selang command.

The last two (DUP and NOTICE) can return only from a selang command.

```
segetstat
```

se_grp_usrs Function

The se_grp_usrs function prints a list of all the users in the group. This is valid only for a group of users not for groups of other classes (for example, GFILE and GHOST).

For example, the following command prints all the users in the group called “system”:

```
se_grp_usrs system
```

se_grp_usrs *group_name*

se_is_running Function

The se_is_running function tells you whether the seosd daemon is currently running. The function returns “yes” if seosd is currently running; otherwise, it returns “no.”

```
se_is_running
```

selang Function

The selang function executes eTrust AC commands. The parameters are transferred as they are to the selang utility. The selang output is returned as the result. This is the only command valid for changing data inside the database.

Note: This function does not actually invoke selang. It uses an API to contact the database or agent, using the same syntax as selang.

For example, the following command returns the properties of the file inside the database if it exists; otherwise, it returns the selang error message:

```
selang sr file /opt/CA/eTrustAccessControl/bin/selang
selang parameters
```

se_objs_in_grp Function

The se_objs_in_grp function prints all the members of a group of resources that belong to the same class (GHOST, GTERMINAL, GFILE, and GSUDO).

For example, the following command prints all the hosts inside the group DevTerms:

```
se_objs_in_grp GHOST DevTerms
se_objs_in_grp class object_list
```

se_scan_props Function

The se_scan_props function prints the properties of an object inside the database. If you do not specify a property, the function returns a list of all the object's properties. You can specify a list of properties by separating them with spaces.

For example, the following command returns the root user audit mode:

```
se_scan_props user root AUDIT_MODE
se_scan_props class object [properties]
```

sewhoami Function

The *sewhoami* function tells you under what current eTrust AC user name you are running (note that this can be different from the current UNIX user).

```
Sewhoami
```

Appendix B: Obsolete API

This section contains the following topics:

The eAC IR API

This library supplies an interface to eTrust AC log files - seos.audit and seos.audit.bak.

This API enables an external application to view audit records in chronological order.

Important! The API described in this appendix is obsolete and is available for backward comparability purposes only. Refer to the current IR API (see page 269) described earlier in this guide.

Structures

Data Position

```
typedef struct
{
        char filename[MAX_PATH];        current log file
        off_t offset;                           offset
        unsigned char md5_1[16];        md5 signature of current log file
        unsigned char md5_2[16];        md5 signature of next log file
} IRDataPosition;
```

UT8 Interpreted Audit Record

```
typedef struct tagSeosUTF8LogInterpreted
{
        char *Label;            Label of an audit field in the record
        char *Value;            Value assigned in the audit record to the field
} SEOS_UTF8AUDLOGINTERP;
```

Possible Return Values

Reason Code	Integer Value	Meaning
IR_EAC_SUCCESS	0	success
IR_EAC_NO_MORE_DATA	1	no more data
IR_EAC_DATA_LOST	2	data lost
IR_EAC_GENERAL_FAIL	3	general failure
IR_EAC_PSBLY_DATA_LOST	4	possibly lost data

Functions

int eacIR_LogInit(IRDataPosition *pos);

Gets the audit filenames (from the eTrust AC settings);

Resets the offset in order to get the oldest log file, and opens it for read - the audit record.

Initializes the IRDataPosition pointer for calling eacIR_LogGetNext().

If the function succeeds, it returns IR_EAC_SUCCESS.

If the function fails, it returns IR_EAC_GENERAL_FAIL

IRDataPosition

(OUT) pointer to IRDataPosition structure.

int eacIR_LogReset(IRDataPosition *pos);

Opens log file (if still exists) according to a given (argument) data position.

Enables you to go back to an old (saved) state and continue from there.

If the function succeeds, it returns IR_EAC_SUCCESS.

If the function fails, it returns IR_EAC_GENERAL_FAIL

IRDataPosition

(IN) pointer to IRDataPosition structure.

int eacIR_LogGetNext(IRDataPosition *pos, SEOS_UTF8AUDLOGINTERP **ppUtfMsg, int *log_type);

Returns the next log record and its type according to a given data position.

Possible return values are:

Condition	Value
Success	IR_EAC_SUCCESS
No more data	IR_EAC_NO_MORE_DATA
data lost	IR_EAC_DATA_LOST
general failure	IR_EAC_GENERAL_FAIL
possibly data lost	IR_EAC_PSBLY_DATA_LOST

IRDataPosition

(IN) pointer to IRDataPosition - represents the position to get the next log record from.

UTF8 strings

(OUT) pointer to (UTF8 strings) interpreted record.

int

(OUT) pointer to int - represents the next log record type.

int eacIR_LogTerminate(void);

Closes log file for read. Call it when finished working with API.

If the function succeeds, it returns IR_EAC_SUCCESS.

If the function fails, it returns IR_EAC_GENERAL_FAIL

void eacIR_LogFreeInterpretRecord(void);

This function frees the last interpreted record.

There are no return values for this function.

Note: eacIR_LogGetNext() API calls the same internal function each time it runs, so you do not need to use this API before calling for next log record.

char *eacIR_LogGetVersion(void);

Returns the API version as a string.

The return value is the version number as a string.

Index