DCE 1.2.2 Application Development Reference

OSF® DCE Product Documentation

The Open Group

OSF® DCE Product Documentation:

DCE 1.2.2 Application Development Reference, (Volume 1)
ISBN 1–85912–103–9
Document Number F205A

DCE 1.2.2 Application Development Reference, (Volume 2)
ISBN 1–85912–108–X
Document Number F205B

DCE 1.2.2 Application Development Reference, (Volume 3)
ISBN 1–85912–159–4
Document Number F205C

Published in the U.K. by The Open Group, 1997.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:
OGPubs@opengroup.org

OTHER NOTICES

Contents

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the IT DialTone. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritizing, and communicating customer requirements to vendors
- conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the Open Brand, represented by the "X" mark, that designates vendor products which conform to Open Group Product Standards
- promoting the benefits of IT DialTone to customers, vendors, and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

The Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product

Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The “X” mark is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the X/Open Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

CAE Specifications

CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our Product Standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

Preliminary Specifications

Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as

stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

Preliminary Specifications are analogous to the trial-use standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

Consortium and Technology Specifications

The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

Product Documentation

This includes product documentation—programmer's guides, user manuals, and so on—relating to the Prestructured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

Guides

These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

Technical Studies

Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They

are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all live documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new Version indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it replaces the previous publication.
- A new Issue indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/public/pubs*.

This Book

The *DCE 1.2.2 Application Development Reference* provides complete and detailed reference information to help application programmers use the correct syntax for Distributed Computing Environment (DCE) calls when writing UNIX applications for a distributed computing environment.

Audience

This document is written for application programmers who want to write Distributed Computing Environment applications for a UNIX environment.

Applicability

This document applies to the OSF ® DCE Version 1.2.2 offering and related updates. See your software license for details.

Purpose

The purpose of this document is to assist application programmers when writing UNIX applications for a distributed computing environment. After reading this manual, application programmers should be able to use the correct syntax for DCE calls.

Document Usage

This document consists of six chapters and is organized into three volumes.

- Volume 1 (Document Number 205A, ISBN 1–85912–103–9) includes:
  - — DCE Routines (Chapter 1)

  - — DCE Threads (Chapter 2)
  - — DCE Remote Procedure Call (beginning of Chapter 3)
- Volume 2 (Document Number 205B, ISBN 1–85912–108–X) includes:
  - — DCE Remote Procedure Call (Chapter 3, continued)
  - — DCE Directory Service (Chapter 4)
  - — DCE Distributed Time Service (Chapter 5)
  - — DCE Security Service (beginning of Chapter 6)
- Volume 3 (Document Number 205C, ISBN 1–85912–159–4) includes:
  - — DCE Security Service (Chapter 6, continued)

Related Documents

For additional information about the Distributed Computing Environment, refer to the following documents:

- *DCE 1.2.2 Introduction to OSF DCE*
  Document Number F201, ISBN 1–85912–182–9
- *DCE 1.2.2 Command Reference*
  Document Number F212, ISBN 1–85912–138–1
- *DCE 1.2.2 Application Development—Introduction and Style Guide*
  Document Number F202, ISBN 1–85912– 187–X
- *DCE 1.2.2 Application Development Guide—Core Components*
  Document Number F203A, ISBN 1–85912–192–6 (Volume 1)
  Document Number F203B, ISBN 1–85912–154–3 (Volume 2)
- *DCE 1.2.2 Application Development Guide—Directory Services*
  Document Number F204, ISBN 1–85912–197–7
- *DCE 1.2.2 Administration Guide—Introduction*
  Document Number F207, ISBN 1–85912–113–6

- *DCE 1.2.2 Administration Guide—Core Components*
  Document Number F208, ISBN 1–85912–118–7
- *DCE 1.2.2 DFS Administration Guide and Reference*
  Document Number F209A, ISBN 1–85912–123–3 (Volume 1)
  Document Number F209B, ISBN 1–85912–128–4 (Volume 2)
- *DCE 1.2.2 GDS Administration Guide and Reference*
  Document Number F211, ISBN 1–85912–133–0
- *DCE 1.2.2 File-Access Administration Guide and Reference*
  Document Number F216, ISBN 1–85912–158–6
- *DCE 1.2.2 File-Access User's Guide*
  Document Number F217, ISBN 1–85912–163–3
- *DCE 1.2.2 Problem Determination Guide*
  Document Number F213A, ISBN 1–85912–143–8 (Volume 1)
  Document Number F213B, ISBN 1–85912–148–9 (Volume 2)
- *DCE 1.2.2 Testing Guide*
  Document Number F215, ISBN 1–85912–153–5
- *DCE 1.2.2 File-Access FVT User's Guide*
  Document Number F210, ISBN 1–85912–189–6
- *DCE 1.2.2 Release Notes*
  Document Number F218, ISBN 1–85912–168–3

Typographic and Keying Conventions

This guide uses the following typographic conventions:

Bold — **Bold** words or characters represent system elements that you must use literally, such as commands, options, and pathnames.

Italic — *Italic* words or characters represent variable values that you must supply. *Italic* type is also used to introduce a new DCE term.

`Constant width` — Examples and information that the system displays appear in `constant width` typeface.

[ ] — Brackets enclose optional items in format and syntax descriptions.

{ } Braces enclose a list from which y ou must choose an item in format and syntax descriptions.

| A vertical bar separates items in a list of choices.

< > Angle brackets enclose the name of a key on the keyboard.

... Horizontal ellipsis points indicate that you can repeat the preceding item one or more times.

This guide uses the following keying conventions:

< Ctrl-*x* **>** or ^ *x*
The notation **< Ctrl-** *x* **>** or ^ *x* followed by the name of a key indicates a control character sequence. For example, **< Ctrl-C>** means that you hold down the control key while pressing **< C>**.

< Return> The notation **< Return>** refers to the key on your terminal or workstation that is labeled with the word Return or Enter, or with a left arrow.

Pathnames of Directories and Files in DCE Documentation

For a list of the pathnames for directories and files referred to in this guide, see the *DCE 1.2.2 Administration Guide—Introduction* and *DCE 1.2.2 Testing Guide*.

Problem Reporting

If you have any problems with the software or vendor-supplied documentation, contact your software vendor’s customer service department. Comments relating to this Open Group document, however, should be sent to the addresses provided on the copyright page.

Trademarks

Motif®, OSF/1®, and UNIX® are registered trademarks and the IT DialTone™, The Open Group™, and the "X Device"™ are trademarks of The Open Group.

DEC, DIGITAL, and ULTRIX are registered trademarks of Digital Equipment Corporation.

DECstation 3100 and DECnet are trademarks of Digital Equipment Corporation.

HP, Hewlett-Packard, and LaserJet are trademarks of Hewlett-Packard Company.

Network Computing System and PasswdEtc are registered trademarks of Hewlett-Packard Company.

AFS, Episode, and Transarc are registered trademarks of the Transarc Corporation.

DFS is a trademark of the Transarc Corporation.

Episode is a registered trademark of the Transarc Corporation.

Ethernet is a registered trademark of Xerox Corporation.

AIX and RISC System/6000 are registered trademarks of International Business Machines Corporation.

IBM is a registered trademark of International Business Machines Corporation.

DIR-X is a trademark of Siemens Nixdorf Informationssysteme AG.

MX300i is a trademark of Siemens Nixdorf Informationssysteme AG.

NFS, Network File System, SunOS and Sun Microsystems are trademarks of Sun Microsystems, Inc.

PostScript is a trademark of Adobe Systems Incorporated.

Microsoft, MS-DOS, and Windows are registered trademarks of Microsoft Corp.

NetWare is a registered trademark of Novell, Inc.

gss_accept_sec_context

Purpose

Establishes a security context between the application and a context acceptor

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_accept_sec_context(
        OM_uint32 *minor_status,
        gss_ctx_id_t *context_handle,
        gss_cred_id_t verifier_cred_handle,
        gss_buffer_t input_token_buffer,
        gss_channel_bindings_t input_chan_bindings,
        gss_name_t *src_name,
        gss_OID *actual_mech_type,
        gss_buffer_t output_token,
        int *ret_flags,
        OM_uint32 *time_rec,
        gss_cred_id_t *delegated_cred_handle);
```

Parameters

Input

verifier_cred_handle

Specifies the credential handle (the identity) claimed by the context acceptor. This is optional information. The credential must be either an **ACCEPT** type credential or a **BOTH** type credential. If you do not specify a credential handle and specify instead **GSS_C_NO_CREDENTIAL**, the application can accept a context under any registered identity. Use the **gssdce_register_acceptor_identity()** routine to register an identity before specifying **GSS_C_NO_CREDENTIAL**.

input_token_buffer

Specifies the token received from the context acceptor.

input_chan_bindings
Specifies bindings supplied by the context initiator.

Allows the context initiator to bind the channel identification information securely to the security context.

Input/Output

context_handle
Specifies a context handle for a new context. The first time the context initiator uses the routine, specify **GSS_C_NO_CONTEXT** to set up a specific context. In subsequent calls, use the value returned by this parameter.

Output

src_name Returns the authenticated name of the context initiator. This information is optional. If the authenticated name is not required, specify NULL.

To deallocate the authenticated name, pass it to the **gss_release_name()** routine.

actual_mech_type
Returns the security mechanism with which the context was established. The security mechanism will be one of the following:

- **GSSDCE_C_OID_DCE_KRBV5_DES** (for DCE security)
- **GSSDCE_C_OID_KRBV5_DES** (for Kerberos Version 5)

output_token Returns a token to pass to the context acceptor. If no token is to be passed to the context acceptor, the routine sets the length field of the returned token buffer to 0 (zero).

ret_flags Returns a bitmask containing six independent flags, each of which requests that the context support a service option. The following symbolic names are provided to correspond to each flag. The symbolic names should be logically ANDed with the value of *ret_flags* to test whether the context supports the service option.

GSS_C_DELEG_FLAG

True	Delegated credentials are available from the *delegated_cred_handle* parameter.
False	No credentials were delegated.

GSS_C_MUTUAL_FLAG

True	The context acceptor requested mutual authentication.
False	The context acceptor did not request mutual authentication.

GSS_C_REPLAY_FLAG

True	Replayed signed or sealed messages will be detected.
False	Replayed messages will not be detected.

GSS_C_SEQUENCE_FLAG

True	Out-of-sequence signed or sealed messages will be detected.
False	Out-of-sequence signed or sealed messages will not be detected.

GSS_C_CONF_FLAG

True	Confidentiality services are available by calling the **gss_seal()** routine.
False	Confidentiality services are not available. However, the application can call the **gss_seal()** routine to provide message encapsulation, data-origin authentication, and integrity services.

GSS_C_INTEG_FLAG

True	Integrity services can be invoked by calling either the **gss_sign()** or **gss_seal()** routine.
False	Integrity services for individual messages are not available.

time_rec Returns the number of seconds for which the context remains valid. This is optional information. If the time is not required, specify NULL.

delegated_cred_handle
Returns the credential handle for credentials received from the context initiator. The credential handle is valid only if delegated

credentials are available. If the *ret_flags* parameter is true, the flag **GSS_C_DELEG_FLAG** is set, indicating that delegated credentials are available.

minor_status Returns a status code from the security mechanism.

Description

The **gss_accept_sec_context()** routine is the second step in establishing a security context between the context initiator and a context acceptor. In the first step, the context initiator calls the **gss_init_sec_context()** routine. The **gss_init_sec_context()** routine generates a token for the security context and passes it to the context initiator. The context initiator sends the token to the context acceptor.

In the second step, the context acceptor accepts the call from the context initiator and calls the **gss_accept_sec_context()** routine. The **gss_accept_sec_context()** routine expects a value for the *input_token* parameter. The value for the *input_token* parameter is generated by the **gss_init_sec_context()** routine and passed by the initiator to the acceptor.

The **gss_accept_sec_context()** routine can also return a value for the *output_token* parameter. The context acceptor presents the token to the **gss_init_sec_context()** routine. If the acceptor does not need to send a token to the initiator, **gss_accept_sec_context()** sets the length field of the *output_token* parameter to 0 (zero).

To complete establishing the context, the context initiator can require one or more reply tokens from the context acceptor. If the application requires reply tokens, the **gss_accept_sec_context()** routine returns a status value containing **GSS_S_CONTINUE_NEEDED**. The application calls the routine again when the reply token is received from the context acceptor. The application passes the token to the **gss_accept_sec_context()** routine via the *output_token* parameters.

The **gss_accept_sec_context()** routine must find a key to decrypt the token. The token contains the unencrypted principal name of the context acceptor. The acceptor's principal name identifies the key that the context initiator used to encrypt the rest of the token. The **gss_accept_sec_context()** routine matches the principal name with the key in the following way:

- If you specify a credential, the credential and the name in the token must match. The acceptor's principal name (contained in the token) has been

registered by a call to the **gssdec_register_acceptor_identity()** routine. The **gss_accept_sec_context()** routine looks in the registered key table.

- If you specify **GSS_C_NO_CRED** and the principal name in the token is registered, the **gss_accept_sec_context()** routine, using either the **rpc_server_register_auth_info()** routine or the **gssdce_register_acceptor_identity()** routine, looks in the table specified when you registered the token name.
- If you specify **GSS_C_NO_CRED** and the principal name in the token is not registered, the **gss_accept_sec_context()** routine fails and returns the status **GSS_S_FAILURE** because the Generic Security Service Application Programming Interface (GSSAPI) does not know where to find the key.

The following table summarizes how the **gss_accept_sec_context()** routine determines the key for the credential:

You specify ...	**Is the principal's name registered?**	**Then the routine ...**
A credential	Yes	Looks in the key table specified in **gssdce_register_acceptor_identity()** or the default key table.
GSS_C_NO_CRED	Yes	Looks in the key table specified in **gssdce_register_acceptor_identity()**.
	No	Fails because the principal is not registered. It returns the status code **GSS_S_FAILURE**.

The values returned using the *src_name*, *ret_flags*, *time_rec*, and *delegated_cred_handle* parameters are not defined unless the routine returns the status **GSS_S_COMPLETE**.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_BINDINGS
The *input_token* parameter contains different channel bindings from those specified with the *input_chan_bindings* parameter.

GSS_S_BAD_SIG
The *input_token* parameter contains an invalid signature.

GSS_S_CONTINUE_NEEDED
To complete the context, the **gss_accept_sec_context( )**routine must be called again with a token required from the context acceptor.

GSS_S_CREDENTIALS_EXPIRED
The referenced credentials have expired.

GSS_S_DEFECTIVE_CREDENTIAL
Consistency checks performed on the credential failed.

GSS_S_DEFECTIVE_TOKEN
Consistency checks performed on the *input_token* parameter failed.

GSS_S_DUPLICATE_TOKEN
The *input_token* parameter was already processed. This is a fatal error that occurs during context establishment.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

GSS_S_NO_CONTEXT
The supplied context handle did not refer to a valid context.

GSS_S_NO_CRED
Indicates either the supplied credentials were not valid for context acceptance or the credential handle did not reference any credentials.

GSS_S_OLD_TOKEN
The *input_token* parameter was too old. This is a fatal error that occurs during context establishment.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gss_delete_sec_context(3sec)**, **gss_init_sec_context(3sec)**, **gssdce_register_acceptor_identity(3sec)**.

gss_acquire_cred

Purpose

Allows an application to acquire a handle for an existing named credential

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_acquire_cred(
        OM_uint32 *minor_status,
        gss_name_t desired_name,
        OM_uint32 time_req,
        gss_OID_set desired_mechs,
        int cred_usage,
        gss_cred_id_t *output_cred_handle,
        gss_OID_set *actual_mechs,
        OM_int32 *time_rec);
```

Parameters

Input

desired_name
Specifies the principal name to use for the credential.

time_req Specifies the number of seconds that credentials remain valid.

desired_mechs
Specifies the object identifier (OID) set for the security mechanism to use with the credential, as follows:

DCE security
Specify **GSS_C_NULL_OID_SET**.

Kerberos
Specify **GSSDCE_C_OID_KRBV5_DES**.

Both DCE security and Kerberos
: Specify **GSSDCE_C_OID_DCE_KRBV5_DES** and **GSSDCE_C_OID_KRBV5_DES**.

To help ensure portability of your application, request the default security mechanism by specifying **GSS_C_NULL_OID_SET**.

cred_usage Specify one of the following:

GSS_C_BOTH
: Specifies credentials that the context initiator can use to either initiate or accept security contexts.

GSS_C_ACCEPT
: Specifies credentials that the context initiator can use only to accept security contexts.

Output

output_cred_handle
: Returns the handle for the return credential.

actual_mechs
: Returns a set of mechanisms for which the credential is valid. This information is optional. If you do not want a set of mechanisms returned, specify NULL.

time_rec Returns the actual number of seconds for which the return credential remains valid. This information is optional. If the actual number of seconds is not required, specify NULL.

minor_status Returns a status code from the security mechanism.

Description

The **gss_acquire_cred( )**routine allows an application to obtain a handle for either an **ACCEPT** or a **BOTH** credential. The application then passes the credential handle to either the **gss_init_sec_context()** routine or the **gss_accept_sec_context()** routine.

Credential handles created by the **gss_acquire_cred()** routine contain a principal name. If the principal name is unregistered, the **gss_acquire_cred()** routine automatically registers the principal in the default key table. You can change the principal's key table by calling the **gssdce_register_acceptor_identify()** routine.

To create an **INITIATE** credential, you must use the **gssdce_login_context_to_cred()** routine.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_MECH
The requested security mechanism is unsupported or unavailable.

GSS_S_BAD_NAMETYPE
The name passed by the *desired_name* parameter is unsupported.

GSS_S_BAD_NAME
An invalid name was passed by the *desired_name* parameter.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

Related Information

Functions: **gssdce_accept_sec_context(3sec)**, **gssdce_create_empty_oid_set(3sec)**, **gssdce_login_context_to_credential(3sec)**, **gssdce_register_acceptor_identity(3sec)**, **gss_init_sec_context(3sec)**.

gss_compare_name

Purpose

Allows an application to compare two internal names to determine whether they are equivalent

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_compare_name(
        OM_uint32 *minor_status,
        gss_name_t name1,
        gss_name_t name2,
        int *name_equal);
```

Parameters

Input

name1 Specifies the first internal name.

name2 Specifies the second internal name.

Output

name_equal Returns one of the following values:

TRUE The names are the same.

FALSE The names are not the same.

minor_status Returns a status code from the security mechanism.

Description

The **gss_compare_name()** routine lets an application compare two internal names to determine whether they are the same. This routine does not resolve the names to see if they refer to the same object. It simply compares the input names for equivalence.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_NAMETYPE
The name passed by the *name1* or *name2* parameter is unsupported.

GSS_S_BAD_NAME
An invalid name was passed by the *name1* or *name2* parameter.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_display_name(3sec)**, **gss_import_name(3sec)**, **gss_release_name(3sec)**.

gss_context_time

Purpose

Checks the number of seconds for which the context will remain valid

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_context_time(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        OM_int32 *time_rec);
```

Parameters

Input

context_handle
: Specifies the context to be checked.

Output

time_rec
: Returns the number of seconds that the context will remain valid. Returns a 0 (zero) if the context has already expired.

minor_status
: Returns a status code from the security mechanism.

Description

The **gss_context_time()** routine checks the number of seconds for which the context will remain valid.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CONTEXT_EXPIRED
The context has already expired.

GSS_S_CREDENTIALS_EXPIRED
The context is recognized but the associated credentials have expired.

GSS_S_NO_CONTEXT
The context identified in the *context_handle* parameter was not valid.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

gss_delete_sec_context

Purpose

Deletes a security context

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_delete_sec_context(
        OM_uint32 *minor_status,
        gss_ctx_id_t *context_handle,
        gss_buffer_t output_token_buffer);
```

Parameters

Input/Output

context_handle
: Specifies the context handle for the context to delete.

Output

minor_status Returns a status code from the security mechanism.

output_token_buffer
: Returns a token to pass to the context acceptor.

Description

The **gss_delete_sec_context()** routine deletes a security context. It also deletes the local data structures associated with the security context. When it deletes the context, the routine can generate a token. The application passes the token to the context acceptor. The context acceptor then passes the token to the **gss_process_context_token()** routine, telling it to delete the context and all associated local data structures.

When the context is deleted, the applications cannot use the *context_handle* parameter for additional security services.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

GSS_S_NO_CONTEXT
The supplied context handle did not refer to a valid context.

Related Information

Functions: **gss_accept_sec_context(3sec)**, **gss_init_sec_context(3sec)**, **gss_process_context_token(3sec)**.

gss_display_name

Purpose

Provides to an application the textual representation of an opaque internal name

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_display_name(
        OM_uint32 *minor_status,
        gss_name_t input_name,
        gss_buffer_t output_name_buffer,
        gss_OID *output_name_type);
```

Parameters

Input

input_name Specifies the name to convert to text.

Output

output_name_buffer
Returns the name as a character string.

output_name_type
Returns the type of name to display as a pointer to static storage. The application should treat this as read-only.

minor_status Returns a status code from the security mechanism.

Description

The **gss_display_name()** routine provides an application with the text form of an opaque internal name. The application can use the text to display the name but not to print it.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_NAMETYPE
The name passed by the *input_name* parameter is recognized.

GSS_S_BAD_NAME
An invalid name was passed by the *input_name* parameter.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_compare_name(3sec)**, **gss_import_name(3sec)**, **gss_release_name(3sec)**.

gss_display_status

Purpose

Provides an application with the textual representation of a GSSAPI status code that can be displayed to a user or used for logging

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_display_status(
        OM_uint32 *minor_status,
        int status_value,
        int status_type,
        gss_OID mech_type,
        int *message_context,
        gss_buffer_t status_string);
```

Parameters

Input

status_value Specifies the status value to convert.

status_type Specifies one of the following status types:

GSS_C_GSS_CODE
Major status; a GSS status code.

GSS_C_MECH_CODE
Minor status; either a DCE security status code or a Kerberos status code.

mech_type Specifies the security mechanism. To use DCE security, specify either of the following:

- **GSSDCE_C_OID_DCE_KRBV5_DES**
- **GSS_C_NULL_OID_SET**

To use Kerberos Version 5, specify **GSSDCE_C_OID_KRBV5_DES**.

Input/Output

message_context
: Indicates whether the status code has multiple messages to read.

  The first time an application calls the routine, you initialize the parameter to 0 (zero). The routine returns the first message. If there are more messages, the routine sets the parameter to a nonzero value. The application calls the routine repeatedly to get the next message, until the *message_context* parameter is zero again.

Output

status_string Returns the status value as a text message.

minor_status Returns a status code from the security mechanism.

Description

The **gss_display_status()** routine provides the context initiator with a textual representation of a Generic Security Service Application Programming Interface (GSSAPI) status code so that the application can display the message to a user or log the message. Because some status values can indicate more than one error, the routine enables the calling application to process status codes with multiple messages.

The *message_context* parameter indicates which error message the application should extract from the *status_value* parameter. The first time an application calls the routine, it should initialize the *message_context* parameter to 0 (zero) and return the first message. If there are additional messages to read, the **gss_display_status()** routine returns a nonzero value. The application can call **gss_display_status()** repeatedly to generate a single text string for each call.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
: The routine was completed successfully.

GSS_S_BAD_MECH
The translation requires a mechanism that is unsupported or unavailable.

GSS_S_BAD_STATUS
Either the status value was not recognized or the status type was something other than **GSS_C_GSS_CODE** or **GSS_C_MECH_CODE**.

GSS_S_FAILURE
The routine failed. Check the *minor_status* for details.

Related Information

Functions: **gss_accept_sec_context(3sec)**, **gss_acquire_cred(3sec)**, **gss_compare_name(3sec)**, **gss_delete_sec_context(3sec)**, **gss_display_status(3sec)**, **gss_import_name(3sec)**, **gss_inquire_cred(3sec)**, **gssdce_extract_creds_from_sec_context(3sec)**, **gssdce_login_context_to_cred(3sec)**.

gss_import_name

Purpose

Converts a printable name to an internal form

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_import_name(
        OM_uint32 *minor_status,
        gss_buffer_t input_buffer_name,
        gss_OID input_name_type,
        gss_name_t *output_name);
```

Parameters

Input

input_name_buffer
: Specifies the buffer containing the printable name to convert.

input_name_type
: Specifies the object identifier for the type of printable name.

  Specify **GSS_C_NULL_OID** to use the DCE name. You can explicitly request the DCE name by using **GSSDCE_C_OID_DCE_NAME**. To help ensure portability of your application, use the default, **GSS_C_NULL_OID**.

Output

output_name Returns the name in an internal form.

minor_status Returns a status code from the security mechanism.

Description

The **gss_import_name()** routine converts a printable name to an internal form.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_NAMETYPE
The name passed by the *input_name* parameter is not recognized.

GSS_S_BAD_NAME
The routine could not interpret the *input_name* parameter as a name of the type specified.

GSS_S_FAILURE
Check the minor status for details.

Related Information

Functions: **gss_compare_name(3sec)**, **gss_display_name(3sec)**, **gss_release_name(3sec)**.

gss_indicate_mechs

Purpose

Allows an application to determine which underlying security mechanisms are available

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_indicate_mechs(
        OM_uint32 *minor_status,
        gss_OID_set *mech_set);
```

Parameters

Output

mech_set Returns the set of supported security mechanisms. The value of **gss_OID_set** is a pointer to a static storage and should be treated as read-only by the context initiator.

minor_status Returns a status code from the security mechanism.

Description

The **gss_indicate_mechs()** routine enables an application to determine which underlying security mechanisms are available. These are DCE security and Kerberos Version 5.

You can use the **gssdce_test_oid_set_member()** routine to check whether a specific security mechanism is available.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gssdce_test_oid_set_member(3sec)**.

gss_init_sec_context

Purpose

Establishes a security context between the context initiator and a context acceptor

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_init_sec_context(
        OM_uint32 *minor_status,
        gss_cred_id_t claimant_cred_handle,
        gss_ctx_id_t *context_handle,
        gss_name_t target_name,
        gss_OID mech_type,
        int req_flags,
        int time_req,
        gss_channel_bindings_t input_channel_bindings,
        gss_buffer_t input_token,
        gss_OID *actual_mech_types,
        gss_buffer_t output_token,
        int *ret_flags,
        OM_int32 *time_rec);
```

Parameters

Input

claimant_cred_handle
: Specifies an optional handle for the credential. To use the default credential, supply **GSS_C_NO_CREDENTIAL**. The credential handle created refers to the DCE default login context. The credential must be either an **INITIATE** or **BOTH** type credential.

target_name
: Specifies the name of the context acceptor.

mech_type
: Specifies the security mechanism. To use DCE security, specify either of the following:

- **GSS_C_OID_DCE_KRBV5_DES**
- **GSS_C_NULL_OID**

To use Kerberos, specify **GSS_C_OID_KRBV5_DES**.

req_flags Specifies four independent flags, each of which requests that the context support a service option. The following symbolic names are provided to correspond to each flag. The symbolic names should be logically ORed to form a bit-mask value.

GSS_C_DELEG_FLAG

TRUE Credentials were delegated to the context acceptor.

FALSE No credentials were delegated.

GSS_C_MUTUAL_FLAG

TRUE The context acceptor has been asked to authenticate itself.

FALSE The context initiator has not been asked to authenticates itself.

GSS_C_REPLAY_FLAG

TRUE Replayed signed or sealed messages will be detected.

FALSE Replayed messages will not be detected.

GSS_C_SEQUENCE_FLAG

TRUE Out-of-sequence signed or sealed messages will be detected.

FALSE Out-of-sequence signed or sealed messages will not be detected.

time_req Specifies the desired number of seconds for which the context should remain valid. To specify the default validity period, use 0 (zero).

input_chan_bindings
Specifies the bindings set by the context initiator. Allows the context initiator to bind the channel identification information securely to the security context.

input_token Specifies the token received from the context acceptor.

The first time the application calls the routine, you specify **GSS_NO_BUFFER**. Subsequent calls require a token from the context acceptor.

Input/Output

context_handle
Specifies the context handle for the new context.

The first time the application calls the routine, you specify **GSS_C_NO_CONTEXT**. Subsequent calls use the value returned by the first call.

Output

actual_mech_type
Returns one of the following values indicating the security mechanism:

- **GSS_C_OID_DCE_KRBV5_DES** for DCE security
- **GSS_C_OID_KRBV5_DES** for Kerberos

output_token Returns the token to send to the context acceptor.

If the length field of the returned buffer is 0 (zero), no token is sent.

ret_flags Returns six independent flags, each of which indicates that the context supports a service option. The following symbolic names are provided to correspond to each flag:

GSS_C_DELEG_FLAG

TRUE	Credentials were delegated to the context acceptor.
FALSE	No credentials were delegated.

GSS_C_MUTUAL_FLAG

TRUE	The context acceptor has been asked to authenticate itself.
FALSE	The context acceptor has not been asked to authenticate itself.

GSS_C_REPLAY_FLAG

TRUE	Replayed signed or sealed messages will be detected.

FALSE — Replayed messages will not be detected.

GSS_C_SEQUENCE_FLAG

TRUE — Out-of-sequence signed or sealed messages will be detected.

FALSE — Out-of-sequence signed or sealed messages will not be detected.

GSS_C_CONF_FLAG

TRUE — Confidentiality service can be invoked by calling the **gss_seal()** routine.

FALSE — No confidentiality service is available. (Confidentiality can be provided using the **gss_seal()** routine, which provides only message encapsulation, data-origin authentication, and integrity services.)

GSS_C_INTEG_FLAG

TRUE — Integrity service can be invoked by calling either the **gss_sign()** or **gss_seal()** routine.

FALSE — Integrity service for individual messages is unavailable.

time_rec — Returns the number of seconds for which the context will be valid. If the mechanism does not support credential expiration, the routine returns the value **GSS_C_INDEFINITE**. If the credential expiration time is not required, specify NULL.

minor_status — Returns a status code from the security mechanism.

Description

The **gss_init_sec_context()** routine is the first step in the establishment of a security context between the context initiator and the context acceptor. To ensure the portability of the application, use its default credential by supplying **GSS_C_NO_CREDENTIAL** to the *claimant_cred_handle* parameter. Specify an explicit credential when the application needs an additional credential; for example, to use delegation.

The first time the application calls the **gss_init_sec_context()** routine, specify the *input_token* parameter as **GSS_NO_BUFFER**. Calls to the routine can return an *output_token* for transfer to the context acceptor. The context acceptor presents the token to the **gss_accept_sec_context()** routine.

If the context initiator does not require a token, **gss_init_sec_context()** sets the length field of the *output_token* argument to 0 (zero).

To complete establishing the context, the calling application can require one or more reply tokens from the context acceptor. If the application requires reply tokens, the **gss_init_sec_context()** routine returns a status value of **GSS_S_CONTINUE_NEEDED**. The application calls the routine again when the reply token is received from the context acceptor and passes the token to the **gss_init_sec_context()** routine via the *input_token* parameter.

The values returned by the *ret_flags* and *time_rec* parameters are not defined unless the routine returns the status **GSS_S_COMPLETE**.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_BINDINGS
The *input_token* parameter contains different channel bindings from those specified with the *input_chan_bindings* parameter.

GSS_S_BAD_NAMETYPE
The *target_name* parameter contains an invalid or unsupported name type.

GSS_S_BAD_NAME
The *target_name* parameter was incorrectly formed.

GSS_S_BAD_SIG
Indicates either that the *input_token* parameter contains an invalid signature or that the *input_token* parameter contains a signature that could not be verified.

GSS_S_CONTINUE_NEEDED
To complete the context, the **gss_init_sec_context()** routine must be called again with a token required from the context acceptor.

GSS_S_CREDENTIALS_EXPIRED
The referenced credentials have expired.

GSS_S_DEFECTIVE_CREDENTIAL
Consistency checks performed on the credential failed.

GSS_S_DEFECTIVE_TOKEN
Consistency checks performed on the *input_token* parameter failed.

GSS_S_DUPLICATE_TOKEN
The *input_token* parameter was already processed. This is a fatal error that occurs during context establishment.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

GSS_S_NO_CONTEXT
The supplied context handle did not refer to a valid context.

GSS_S_OLD_TOKEN
The *input_token* parameter was too old. This is a fatal error that occurs during context establishment.

Related Information

Functions: **gss_accept_sec_context(3sec)**, **gss_delete_sec_context(3sec)**.

gss_inquire_cred

Purpose

Provides the calling application information about a credential

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_inquire_cred(
        OM_uint32 *minor_status,
        gss_cred_id_t cred_handle,
        gss_name_t *name,
        OM_uint32 *lifetime,
        int *cred_usage,
        gss_OID_set *mechs);
```

Parameters

Input

cred_handle Specifies a handle for the target credential. To get information about the default credential, specify **GSS_C_NO_CREDENTIAL**.

Output

name Returns the principal name asserted by the credential. If the principal name is not required, specify NULL.

lifetime Returns the number of seconds for which the credential will remain valid.

If the credential expired, the parameter returns a 0 (zero). If there is no credential expiration, the parameter returns the value **GSS_C_INDEFINITE**. If an expiration time is not required, specify NULL.

cred_usage Returns one of the following values describing how the application can use the credential:

- **GSS_C_INITIATE**
- **GSS_C_ACCEPT**
- **GSS_C_BOTH**

If no usage information is required, specify NULL.

mechs Returns a set of security mechanisms supported by the credential, as follows:

- **GSSDCE_C_OID_DCE_KRBV5_DES** (for DCE security)
- **GSSDCE_C_OID_KRBV5_DES** (for Kerberos)

minor_status Returns a status code from the security mechanism.

Description

The **gss_inquire_cred()** routine provides information about a credential to the calling application. The calling application must first have called the **gss_acquire_cred()** routine for a handle for the credential.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CREDENTIALS_EXPIRED
The credentials expired. If the *lifetime* parameter was passed as NULL, it is set to 0 (zero).

GSS_S_DEFECTIVE_CREDENTIAL
The credentials were invalid.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CRED
The routine could not access the credentials.

Related Information

Functions: **gss_acquire_cred(3sec)**.

gss_process_context_token

Purpose

Passes a context to the security service

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_process_context_token(
        OM_uint32 *minor_status,
        gss_ctx_id_t *context_handle,
        gss_buffer_t input_token_buffer);
```

Parameters

Input

context_handle
: Specifies the context handle on which the security service processes the token.

input_token_buffer
: Specifies an opaque pointer to the first byte of the token to be processed.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gss_process_context_token()** routine passes tokens generated by the **gss_delete_security_context()** routine to the security service.

Usually, tokens are associated with either the context establishment or with per-message security services. If the tokens are associated with the context establishment, they are passed to the **gss_init_sec_context()** or **gss_accept_sec_context()** routine. If the tokens are associated with the per-message security service, they

are passed to the **gss_verify()** or **gss_unseal()** routine. Tokens generated by the **gss_delete_security_context()** routine are passed by the **gss_process_context_token()** routine to the security service for processing.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_DEFECTIVE_TOKEN
Consistency checks performed on the *input_token* parameter failed.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter return value for more information.

GSS_S_NO_CONTEXT
The supplied context handle did not refer to a valid context.

Related Information

Functions: **gss_delete_security_context(3sec)**.

gss_release_buffer

Purpose

Frees storage associated with a buffer

Synopsis

#include <dce/gssapi.h>

OM_uint32 gss_release_buffer(
OM_uint32 ***_minor_status_,**
gss_buffer_t *buffer***);**

Parameters

Input

buffer The buffer to delete.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gss_release_buffer()** routine deletes the buffer by freeing the storage associated with it.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. See the *minor_status* parameter for details.

gss_release_cred

Purpose

Marks a credential for deletion

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_release_cred(
        OM_uint32 *minor_status,
        gss_cred_id_t *cred_handle);
```

Parameters

Input

cred_handle Specifies the buffer containing the opaque credential handle. This information is optional. To release the default credential, specify **GSS_C_NO_CREDENTIAL**.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gss_release_cred()** routine informs the GSSAPI that a credential is no longer required and marks it for deletion.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CRED
The credentials could not be accessed.

gss_release_name

Purpose

Frees storage associated with an internal name that was allocated by a GSSAPI routine.

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_release_name(
        OM_uint32 *minor_status,
        gss_name_t *name);
```

Parameters

Input

name The name to delete.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gss_release_name()** routine deletes the internal name by freeing the storage associated with that internal name.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_NAME
The *name* parameter did not contain a valid name.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_compare_name(3sec)**, **gss_display_name(3sec)**, **gss_import_name(3sec)**.

gss_release_oid_set

Purpose

Frees storage associated with a **gss_OID_set** object

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_release_oid_set (
        OM_uint32 *minor_status,
        gss_OID_set set);
```

Parameters

Input

set The OID set to delete.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gss_release_oid_set()** routine frees storage that is associated with the **gss_OID_set** parameter and was allocated by a GSSAPI routine.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

gss_seal

Purpose

Cryptographically signs and optionally encrypts a message

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_seal(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        int conf_req_flag,
        int qop_req,
        gss_buffer_t input_message_buffer,
        int *conf_state,
        gss_buffer_t output_message_buffer);
```

Parameters

Input

context_handle
Specifies the context on which the message is sent.

conf_req_flag
Specifies the requested level of confidentiality and integrity services, as follows:

TRUE	Both confidentiality and integrity services are requested.
FALSE	Only integrity services are requested.

qop_req Specifies the cryptographic algorithm, or quality of protection. Specify **GSS_C_QOP_DEFAULT**. The DCE GSSAPI supports only one quality of protection.

input_message_buffer
Specifies the message to seal.

Output

conf_state Returns the requested level of confidentiality and integrity services, as follows:

TRUE	Confidentiality, data origin, authentification, and integrity services have been applied.
FALSE	Only integrity and data origin services have been applied.

output_message_buffer
Returns the buffer to receive the sealed message.

minor_status Returns a status code from the security mechanism.

Description

The **gss_seal()** routine cryptographically signs and optionally encrypts a message. The *output_message* parameter contains both the signature and the message.

Although the *qop_req* parameter enables a choice between several qualities of protection, DCE GSSAPI supports only one quality of protection. If you specify an unsupported protection, the **gss_seal()** routine returns a status of **GSS_S_FAILURE**.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CONTEXT_EXPIRED
The context has already expired.

GSS_S_CREDENTIALS_EXPIRED
The context is recognized but the associated credentials have expired.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CONTEXT
The context identified in the *context_handle* parameter was not valid.

gss_sign

Purpose

Generates a cryptographic signature for a message

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_sign(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        int qop_req,
        gss_buffer_t message_buffer,
        gss_buffer_t msg_token);
```

Parameters

Input

context_handle
: Specifies the context on which the message is sent.

qop_req
: Specifies the cryptographic algorithm, or quality of protection. Specify **GSS_C_QOP_DEFAULT**. DCE GSSAPI supports only one quality of protection.

message_buffer
: Specifies the message to send.

Output

msg_token
: Returns the buffer to receive the signature token to transfer to the context acceptor.

minor_status
: Returns a status code from the security mechanism.

Description

The **gss_sign()** routine generates an encrypted signature for a message. It places the signature in a token for transfer to the context acceptor.

Although the *qop_req* parameter enables a choice between several qualities of protection, DCE GSSAPI supports only one quality of protection. If you specify an unsupported protection, the **gss_sign()** routine returns a status of **GSS_S_FAILURE**.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CONTEXT_EXPIRED
The context has already expired.

GSS_S_CREDENTIALS_EXPIRED
The context is recognized but the associated credentials have expired.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CONTEXT
The context identified in the *context_handle* parameter was not valid.

gss_unseal

Purpose

Converts a sealed message into a usable form and verifies the embedded signature

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_unseal(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        gss_buffer_t input_message_buffer,
        gss_buffer_t output_message_buffer,
        int *conf_state,
        int *qop_state);
```

Parameters

Input

context_handle
: Specifies the context on which the message arrived.

input_message_buffer
: Specifies the sealed message.

output_message_buffer
: Specifies the buffer to receive the unsealed message.

Output

conf_state
: Returns the requested level of confidentiality and integrity services, as follows:

TRUE	Both confidentiality and integrity services are requested.
FALSE	Only integrity services are requested.

qop_state
: Returns the cryptographic algorithm, or quality of protection.

minor_status Returns a status code from the security mechanism.

Description

The **gss_unseal()** routine converts a sealed message to a usable form and verifies the embedded signature. The *conf_state* parameter indicates whether the message was encrypted. The *qop_state* parameter indicates the quality of protection.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_BAD_SIG
The signature was incorrect.

GSS_S_CONTEXT_EXPIRED
The context has already expired.

GSS_S_CREDENTIALS_EXPIRED
The context is recognized but the associated credentials have expired.

GSS_S_DEFECTIVE_TOKEN
The token failed consistency checks.

GSS_S_DUPLICATE_TOKEN
The token was valid and contained the correct signature but it had already been processed.

GSS_S_FAILURE
The routine failed. The context specified in the *context_handle* parameter was not valid.

GSS_S_NO_CONTEXT
The context identified in the *context_handle* parameter was not valid.

GSS_S_OLD_TOKEN
The token was valid and contained the correct signature but it is too old.

GSS_S_UNSEQ_TOKEN
The token was valid and contained the correct signature but it has been verified out of sequence. An earlier token signed or sealed by the remote application has not been processed locally.

Related Information

Functions: **gss_seal(3sec)**, **gss_sign(3sec)**.

gss_verify

Purpose

Checks that the cryptographic signature fits the supplied message

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_verify(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        gss_buffer_t message_buffer,
        gss_buffer_t token_buffer,
        int qop_state);
```

Parameters

Input

context_handle
: Specifies the context on which the message arrived.

message_buffer
: Specifies the message to be verified.

token_buffer
: Specifies the signature token to be associated with the message.

Output

qop_state Returns the cryptographic algorithm, or quality of protection, from the signature.

minor_status Returns a status code from the security mechanism.

Description

The **gss_verify()** routine checks that an encrypted signature, in the *token_buffer* parameter, fits the message in the *message_buffer* buffer. The application receiving the message can use the *qop_state* parameter to check the message's protection.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CONTEXT_EXPIRED
The context has already expired.

GSS_S_CREDENTIALS_EXPIRED
The context is recognized but the associated credentials have expired.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CONTEXT
The context identified in the *context_handle* parameter was not valid.

Related Information

Functions: **gss_seal(3sec)**, **gss_sign(3sec)**.

gssdce_add_oid_set_member

Purpose

Adds an OID to an OID set

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_add_oid_set_member(
        OM_uint32* minor_status,
        gss_OID* member_OID,
        gss_OID_set* OID_set);
```

Parameters

Input

member_OID
: Specifies the OID you want to add to the OID set.

OID_set
: Specifies an OID set.

Output

minor_status
: Returns a status code from the security mechanism.

Description

The **gssdce_add_oid_set_member()** routine adds a new OID to an OID set. If an OID set does not exist, you can create a new, empty OID set with the **gssdce_create_empty_oid_set()** routine.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gssdce_create_empty_oid_set(3sec)**.

gssdce_create_empty_oid_set

Purpose

Creates a new, empty OID set to which members can be added by calling **gssdce_add_oid_set_member()**

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_create_empty_oid_set(
        OM_uint32 *minor_status,
        gss_OID_set *OID_set);
```

Parameters

Input

OID_set Specifies the OID set you want to create.

Output

minor_status Returns a status code from the security mechanism.

Description

The **gssdce_create_empty_oid_set()** routine creates a new, empty OID set to which the context initiator can add members. Use the **gssdce_add_oid_set_member()** routine to add members to the OID set.

Use the **gssdce_create_empty_oid_set()** routine to specify a set of security mechanisms with which you can use an acquired credential. To create a credential that can accept a security context using DCE security, Kerberos, or a combination of the two, use the **gss_acquire_cred()** routine.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gssdce_add_oid_set_member(3sec)**.

gssdce_cred_to_login_context

Purpose

Obtains the DCE login context associated with a GSSAPI credential

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_cred_to_login_context(
        OM_uint32 *minor_status,
        cred_id_t *cred_handle,
        sec_login_handle_t login_context);
```

Parameters

Input

cred_handle
: Specifies the credential handle.

Output

login_context
: Returns the DCE login context associated with the credential.

minor_status Returns a status code from the security mechanism.

Description

Using the **gssdce_cred_to_login_context()** routine, an application can obtain the DCE login context associated with a GSSAPI credential. Only credentials with usage-types **INIT** or **BOTH** have associated login contexts.

Use this routine in the following situations:

- If you want to add delegation notes to a login context
- To use an **INITIATE** or **BOTH** credential to initiate an authenticated RPC call

The application must delete the login context when it no longer needs the credentials or the login context.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_CREDENTIALS_EXPIRED
The credentials have expired.

GSS_S_DEFECTIVE_CREDENTIAL
The credential is defective in some way.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CRED
The routine requested the default login context, but no default login context was available.

Related Information

Functions: **gssdce_login_context_to_cred(3sec)**, **sec_login_purge_contexts(3sec)**, **sec_login_release_context(3sec)**.

gssdce_extract_creds_from_sec_context

Purpose

Extracts a DCE credential from a GSSAPI security context

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_extract_creds_from_sec_context(
        OM_uint32 *minor_status,
        gss_ctx_id_t context_handle,
        rpc_authz_cred_handle_t output_cred);
```

Parameters

Input

context_handle
: Specifies the handle of the security context containing the DCE credential.

Output

output_cred Returns the DCE credential.

minor_status Returns a status code from the security mechanism.

Description

The **gssdce_extract_creds_from_sec_context()** routine extracts the context initiator's DCE credential from a context acceptor's security context. Use this routine if the underlying mechanism type is DCE security (**GSSDCE_C_OID_DCE_KRBV5_DES**).

The context acceptor calls the **gssdce_extract_creds_from_sec_context()** routine to get the DCE credential containing the privilege attributes of the context initiator. DCE

credentials are used by DCE access control list (ACL) managers to determine whether the initiator has the right to access the object to which an ACL refers.

The principal contained in the DCE credential may not be the same as the *src_name* parameter value from the **gss_accept_sec_context()** routine. The principal in the DCE credential may be a compound principal.

If the context was established by calling the **gss_init_set_context()** routine and specifying **GSSDCE_C_OID_KRBV5_DES** to use Kerberos (instead of DCE security), the **gssdce_extract_creds_from_sec_context()** routine returns a major status of O and a minor status of O.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

GSS_S_NO_CONTEXT
The routine could not access the security context.

Related Information

Functions: **gss_init_sec_context(3sec)**.

gssdce_login_context_to_cred

Purpose

Creates a GSSAPI credential handle for a context initiator or context acceptor from a DCE login context

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_login_context_to_cred(
        OM_uint32 *minor_status,
        sec_login_handle_t login_context,
        OM_uint32 lifetime_req,
        OID_set desired_mechs,
        cred_id_t *output_cred_handle,
        OID_set *actual_mechs,
        OM_uint32 lifetime_rec);
```

Parameters

Input

login_context

Specifies the DCE login context handle. To use the default login context handle, specify NULL.

lifetime_req

Specifies the number of seconds that the credential should remain valid.

desired_mechs

Specifies the object identifier (OID) set for the security mechanism to use with the credential, as follows:

DCE security
: Specify **GSS_C_NULL_OID_SET**.

Kerberos
: Specify **GSSDCE_C_OID_KRBV5_DES**.

Both DCE security and Kerberos
: Specify **GSSDCE_C_OID_DCE_KRBV5_DES** and **GSSDCE_C_OID_KRBV5_DES**.

To help ensure portability of your application, use the default security mechanism by specifying **GSS_C_NULL_OID_SET**.

Output

output_cred_handle
: Returns the credential handle.

actual_mechs
: Returns the set specifying the security mechanisms with which the credential can be used. The set can contain one or both of the following:

- **GSSDCE_C_OID_DCE_KRBV5_DES** (for DCE security)
- **GSSDCE_C_OID_KRBV5_DES** (for Kerberos)

lifetime_rec
: Returns the number of seconds that the credential will remain valid.

minor_status Returns a status code from the security mechanism.

Description

The **gssdce_login_context_to_cred()** routine creates a generic security service application programming interface (GSSAPI) credential handle for the context initiator or context acceptor from a DCE login context. The routine creates a credential that can be used to initiate or acquire a security context. Use this routine if you need to create a GSSAPI credential for delegation.

Status Codes

The following describes a partial list of codes (messages) that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all messages. The following status codes can be returned:

GSS_S_COMPLETE
: The routine was completed successfully.

GSS_S_DEFECTIVE_CREDENTIAL
The credential is defective in some way.

GSS_S_NO_CRED
The routine requested the default login context, but no default login context was available.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gssdce_cred_to_login_context(3sec)**.

gssdce_register_acceptor_identity

Purpose

Registers a context acceptor's identity

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gss_register_acceptor_indentity(
        OM_uint32 *minor_status,
        unsigned_char_t *acceptor_principal_name,
        rpc_auth_key_retrieval_fn_t get_key_fn,
        void *arg);
```

Parameters

Input

acceptor_principal_name
: Specifies the principal name to use for the context acceptor.

get_key_fn
: Specifies either the DCE default key-retrieval routine or the address of a routine that returns encryption keys.

arg
: Specifies an argument to pass to the *get_key_fn* key acquisition routine. To specify the DCE default, use NULL.

Output

minor_status
: Returns a status code from the security mechanism.

Description

The **gssdce_register_acceptor_identity()** routine registers the server principal name as an identity claimed by the context acceptor and informs DCE security where to find the key table containing the principal's key information.

The **gssdce_register_acceptor_identity()** routine uses the *get_key_fn* and *arg* parameters of the **rpc_server_register_auth_info()** routine to find the key for the token for the context acceptor's principal name. The following table lists the values for the parameters and which key tables they point to:

Retrieval Routine	Key Table	Explanation
NULL	NULL	Uses the default DCE retrieval routine to get the key from the DCE key table. This is accomplished via the default key table, **/krb/v5srvtab**.
NULL	*string=key_table_name*	Uses the default DCE retrieval routine to get the key from the a key table whose name you specify using the argument string.
routine_address	*user_written_routine*	Uses a user-written retrieval routine to get the key from a key table specified in the routine.

For more information on registering a server with DCE, refer to the **rpc_server_register_auth_info(3rpc)** reference page.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the minor status for details.

Related Information

Functions: **gss_accept_sec_context(3sec)**, **rpc_server_register_auth_info(3rpc)**.

gssdce_set_cred_context_ownership

Purpose

Changes the ownership of a DCE credential's login context

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_set_cred_context_ownership(
        OM_uint32 *minor_status,
        gss_cred_id_t credential_handle,
        int ownership);
```

Parameters

Input

credential_handle
: Specifies the handle of the DCE credential to be modified.

ownership
: Specifies the owner of the DCE credential. Specify one of the following:

GSSDCE_C_OWNWERSHIP_GSSAPI
: Specifies that the credential's login context is owned by the generic security service application programming interface (GSSAPI).

GSSDCE_C_OWNERSHIP_APPLICATION
: Specifies that the credential's login context is owned by the application.

Output

minor_status
: Returns a status code from the security mechanism.

Description

The **gssdce_set_cred_context_ownership()** routine modifies the ownership of a DCE credential's login context. **INIT** type and **BOTH** type credentials have DCE login contexts. Normally, these internal login contexts are deleted when the credential is released (when the application calls the **gss_release_cred()** routine). However, for credentials created by the **gssdce_cred_to_login_context()** and credentials passsed to the **gsscdce_cred_to_login_context()**routine, the application may have an external reference to the credential's login context and may still be using the login context. The GSSAPI will not delete internal login contexts of these credentials when they are released.

This routine allows the application to modify the ownership of a credential's login context. If ownership is changed to **GSSDCE_C_OWNERSHIP_GSSAPI**, the login context is deleted when GSSAPI releases the credential. If ownership is changed to **GSSDCE_C_OWNERSHIP_APPLICATION**, the application is responsible for deleting the login context. DCE credential login contexts that are owned by an application must not be deleted until the credential is released since the GSSAPI may still need to access the credential's login context.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gss_release_buffer(3sec)**, **gssdce_cred_to_login_context(3sec)**.

gssdce_test_oid_set_member

Purpose

Checks an OID set to see if a specified OID is in the set

Synopsis

```
#include <dce/gssapi.h>

OM_uint32 gssdce_test_oid_set_member(
        OM_uint32 *minor_status,
        gss_OID member_OID,
        gss_OID_set set,
        int* is_present);
```

Parameters

Input

member_OID
: Specifies the OID to search for in the OID set.

set
: Specifies the OID set to check.

Output

is_present
: Returns one of the following values to indicate whether the OID is a member of the OID set:

Returns...	If...
1	The OID is present as a member of the OID set
0	The OID is absent, not a member of the OID set

minor_status
: Returns a status code from the security mechanism.

Description

The **gssdce_test_oid_set_member()** routine checks an OID set to see if the specified OID is a member of the set. To add a member to an OID set, use the **gssdce_add_oid_set_member()** routine.

The **gssdce_test_oid_set_member()** routine uses the value of the *actual_mechs* output parameter from the **gss_acquire_cred()** routine to get the list of OIDs. It checks this list to see if any of the OIDs are members of the OID set.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

GSS_S_COMPLETE
The routine was completed successfully.

GSS_S_FAILURE
The routine failed. Check the *minor_status* parameter for details.

Related Information

Functions: **gss_acquire_cred(3sec)**, **gss_indicate_mechs(3sec)**, **gssdce_add_oid_set_member(3sec)**.

pkc_add_trusted_key

Purpose

Adds a key to specified trust list

Synopsis

```
#include <pkc_certs.h>

pkc_add_trusted_key(
        pkc_trust_list_t * trust_list,
        const pkc_trusted_key_t & key);
```

Parameters

Input

trust_list Specifies trust list to which key should be added.

key Specifies key to add.

Description

pkc_add_trusted_key(3sec) adds a specified key to a specified trust list.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_check_cert_against_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_append_to_trustlist

Purpose

Appends one or more items to trust list

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_append_to_trustlist(
        trustlist_t ** tr_list,
        trustitem_t * tr_item,
        size_t no_of_tr_items);
```

Parameters

Input

tr_list Specifies trust list to which item(s) are to be appended.

tr_item Specifies item(s) to append to trust list.

no_of_tr_items
Specifies number of items to append.

Description

pkc_append_to_trustlist(3sec) appends one or more items to a trust list (a **pkc_trust_list_t**, pointed to by **(*tr_list)->handle**).

If the trust list is invalid, **pkc_s_bad_param** is returned.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_ca_key_usage.class

Purpose

A class that expresses key usage

Member Functions

Public

- **pkc_ca_key_usage_t & operator = (unsigned long c)**
- **pkc_ca_key_usage_t(unsigned long c = 0xffffffflu)**

Description

pkc_ca_key_usage_t expresses the usage of a key.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Parent Class

This class is derived from **pkc_generic_key_usage_t**.

Related Information

Classes: **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_check_cert_against_trustlist

Purpose

Checks specified certificate against specified list of trusted keys

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_check_cert_against_trustlist(
        pkc_trust_list_t * trust_list,
        const Certificate * cert,
        int revoked_certs_permitted);
```

Parameters

Input

trust_list Specifies list of trusted keys to check certificate against.

cert Specifies certificate to check.

revoked_certs_permitted
Specifies whether revoked certificates should still be trusted for dates prior to their revocation date.

Description

pkc_check_cert_against_trustlist(3sec) checks the specified certificate against the specified list of trusted keys. If the certificate is valid and can be verified from the trust list, its content is added to the trust list. *revoked_certs_permitted* is a flag that specifies whether revoked certificates should still be trusted for dates prior to their revocation date.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_constraints.class

Purpose A class that expresses constraints on names

Member Data

Public

- **unsigned path_length**

  The maximum path length that can be certified by the key (if the entity can act as a certifying authority). **0xffffu** means ‘‘unlimited’’.

- **pkc_name_subord_constraints_t subord_constraints**
- **pkc_name_subtree_constraints_t subtree_constraints**

Member Functions

Public

- **pkc_constraints_t & operator = (const pkc_constraints_t & o)**
- **pkc_constraints_t(void)**
- **unsigned32 constrain()**

  Adds the specified constraints. Takes the following argument:

  — **const pkc_constraints_t & o**

- **char is_permitted() const**

  Takes the following arguments:

  — **const x500name & ca_name**

  — **const x500name & subject_name**

- **void get_next_link_constraint() const**

  Generates a new name constraint that will be applicable to a certificate issued by the subject of this constraint. Takes the following argument:

— **pkc_constraints_t ** new_constraints**

Description

pkc_constraints_t is a class that expresses constraints on the names that can be certified by a given key. Three types of constraint can be checked: total path length, name subordination, and subtree constraints.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_copy_trustlist

Purpose

Copies a trustlist

Synopsis

#include <pkc_certs.h>

unsigned32 pkc_copy_trustlist(
const pkc_trust_list_t * *input_trust_list***,**
pkc_trust_list_t * *output_trust_list***);**

Parameters

Input

input_trust_list
The trust list to be copied.

Output

output_trust_list
The copied trust list.

Description

pkc_copy_trustlist(3sec) creates a functionally equivalent copy of a trust list. The key ids within the newly created trust list will be different from those in the original trust list, but the keys and the trust relationships between them will be the same.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_delete_trustlist(3sec)**, **pkc_display_trustlist(3sec)**, **pkc_lookup_element_in_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_crypto_generate_keypair

Purpose

Generates a pair of public and private keys

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_crypto_generate_keypair(
        gss_OID algorithm,
        unsigned32 size,
        void * alg_info,
        sec_pk_data_t * private_key,
        sec_pk_data_t * public_key);
```

Parameters

Input

algorithm Specifies the crypto module.

size Specifies the key size.

alg_info Specifies algorithm-specific information, if any.

Output

private_key The generated private key.

public_key The generated public key.

Description

pkc_crypto_generate_keypair generates a pair of public and private keys. The **(*generate_keypair)()** routine of the crypto module specified by *algorithm* is called to do this (but note that crypto modules are not required to provide a **(*generate_keypair)()** function).

The *size* parameter will be used by the routine to determine the key size in some way defined by the algorithm; for the RSA algorithm, for example, it should be treated as the number of bits in the key modulus. The *private_key* and *public_key* parameters should be expected to return BER-encoded **PrivateKeyInfo** and **SubjectPublicKeyInfo** data objects respectively.

The *alg_info* parameter can be used for algorithm-specific information to modify the key generation process; NULL can be specified.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_intro(3sec)**, **pkc_crypto_get_registered_algorithms(3sec)**, **pkc_crypto_lookup_algorithm(3sec)**, **pkc_crypto_register_signature_alg(3sec)**, **pkc_crypto_sign(3sec)**, **pkc_crypto_verify_signature(3sec)**.

pkc_crypto_get_registered_algorithms

Purpose

Returns algorithm implementations

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_crypto_reg.h>

pkc_crypto_get_registered_algorithms(
        gss_OID_set *oid_set);
```

Parameters

Output

oid_set A pointer to an OID set describing the currently registered algorithm implementations.

Description

pkc_crypto_get_registered_algorithms(3sec) returns an OID set describing the currently registered algorithm implementations.

pkc_crypto_lookup_algorithm(3sec) may be called to obtain details about a particular algorithm.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_generate_keypair(3sec)**, **pkc_crypto_lookup_algorithm(3sec)**, **pkc_crypto_register_signature_alg(3sec)**, **pkc_crypto_sign(3sec)**, **pkc_crypto_verify_signature(3sec)**.

pkc_crypto_lookup_algorithm

Purpose

Returns cryptographic module details

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_crypto_reg.h>

unsigned32 pkc_crypto_lookup_algorithm(
        gss_OID oid,
        pkc_signature_algorithm_t *details);
```

Parameters

Input

oid An OID identifying the algorithm about which details are desired.

Output

details A pointer to an algorithm implementation descriptor block for the specified algorithm.

Description

pkc_crypto_lookup_algorithm(3sec) returns a pointer to an algorithm implementation descriptor block for the specified cryptographic algorithm, and leaves the algorithm list unlocked. Calling this routine is the recommended way of obtaining information about a registered algorithm implementation.

The complete list of registered algorithms may be obtained by calling **pkc_crypto_get_registered_algorithms(3sec)**.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_generate_keypair(3sec)**, **pkc_crypto_get_registered_algorithms(3sec)**, **pkc_crypto_register_signature_alg(3sec)**, **pkc_crypto_sign(3sec)**, **pkc_crypto_verify_signature(3sec)**.

pkc_crypto_register_signature_alg

Purpose

Registers a signature algorithm module

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_crypto_reg.h>

unsigned32 pkc_crypto_register_signature_alg(
        pkc_signature_algorithm_t * alg,
        int replacement_policy);
```

Parameters

Input

alg A pointer to the signature algorithm module structure to be registered.

replacement_policy
Specifies how the registration is to be handled if an implementation of the algorithm is already registered. There are three possible values:

PKC_REPLACE_NONE
Specifies that an error should be returned if an implementation of the algorithm is already registered.

PKC_REPLACE_ENTRYPOINTS
Specifies that only entry points that the original implementation (if any) did not provide should be replaced. (Note that this value is not currently supported.)

PKC_REPLACE_ALL
Specifies that the new implementation should replace the existing one, if any.

pkc_crypto_register_signature_alg(3sec)

Description

pkc_crypto_register_signature_alg(3sec) registers a signature algorithm module, in the form of a properly declared **pkc_signature_algorithm_t** data structure, which contains identifying information about the module as well as entry points to all of the module's functions.

Calling this routine will cause the module passed to it to be registered as a cryptographic module; it can then be accessed by other applications via the high level certification API routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_generate_keypair(3sec)**, **pkc_crypto_get_registered_algorithms(3sec)**, **pkc_crypto_lookup_algorithm(3sec)**, **pkc_crypto_sign(3sec)**, **pkc_crypto_verify_signature(3sec)**.

pkc_crypto_sign

Purpose

Signs data with private key

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_crypto_reg.h>

pkc_crypto_sign(
        gss_OID algorithm,
        sec_pk_gen_data_t data,
        sec_pk_data_t private_key,
        sec_pk_data_t *signature);
```

Parameters

Input

algorithm An OID identifying the cryptographic algorithm to be used in signing the data.

data The data to be signed.

private_key The private key (i.e., private member of a public-private key pair) to be used to sign the data.

Output

signature The signature generated by the algorithm on the data passed.

Description

pkc_crypto_sign(3sec) searches the list of registered algorithms for an implementation of the specified algorithm. If found, the implementation is opened, if necessary, and its **(sign)()** function invoked to sign the *data*. The *signature* is returned to the caller.

Using this routine, an application can get data signed in one simple call. The alternative is to lookup the desired cryptographic module by calling **pkc_crypto_lookup_algorithm(3sec)**, then explicitly call the module's **(sign)()** routine.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_generate_keypair(3sec)**, **pkc_crypto_get_registered_algorithms(3sec)**, **pkc_crypto_lookup_algorithm(3sec)**, **pkc_crypto_register_signature_alg(3sec)**, **pkc_crypto_verify_signature(3sec)**.

pkc_crypto_verify_signature

Purpose

Verifies a signature

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_crypto_reg.h>

pkc_crypto_verify_signature(
        gss_OID algorithm,
        sec_pk_gen_data_t data,
        sec_pk_data_t public_key,
        sec_pk_data_t signature);
```

Parameters

Input

algorithm An OID identifying the cryptographic algorithm to be used in verifying the data.

data The signed data whose signature is to be verified.

public_key The public key (i.e., public member of a public-private key pair) to be used to verify the signed data.

signature The signature to be verified.

Description

pkc_crypto_verify_signature(3sec) searches the list of registered algorithms for an implementation of the specified algorithm. If found, the implementation is opened, if necessary, and its **(verify)()** function invoked to verify the data and signature passed by the caller.

The routine returns 0 for a correct signature, **pkc_invalid_signature** for an incorrect signature, or another DCE-defined error status to indicate any other errors.

Using this routine, an application can verify signed data in one simple call. The alternative is to lookup the desired cryptographic module by calling **pkc_crypto_lookup_algorithm(3sec)**, then explicitly call the module's **(verify)()** routine.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_crypto_generate_keypair(3sec)**, **pkc_crypto_get_registered_algorithms(3sec)**, **pkc_crypto_lookup_algorithm(3sec)**, **pkc_crypto_register_signature_alg(3sec)**, **pkc_crypto_sign(3sec)**.

pkc_delete_trustlist

Purpose

Deletes a trust list

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_delete_trustlist(
        pkc_trust_list_t * trust_list);
```

Parameters

Input

trust_list The trust list to be deleted.

Description

pkc_delete_trustlist(3sec) deletes a trust list and all keys within it.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_copy_trustlist(3sec)**, **pkc_display_trustlist(3sec)**, **pkc_lookup_element_in_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_display_trustlist

Purpose

Displays information about a trust list

Synopsis

#include <pkc_certs.h>

unsigned32 pkc_display_trustlist(
const pkc_trust_list_t * *input_trust_list***);**

Parameters

Input

input_trust_list
The trust list to be displayed.

Description

pkc_display_trustlist(3sec) is a debugging routine, intended for use when developing a policy module. It prints information about a trust list (all the keys, the trust relationships between keys, etc.) to standard output. It can be used to verify that the trust chains that a policy module implementor expects are actually being built within the trust list.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_copy_trustlist(3sec)**, **pkc_delete_trustlist(3sec)**, **pkc_lookup_element_in_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_free

Purpose

Frees storage allocated by certification routines

Synopsis

```
#include <pkc_api.h>

extern void pkc_free(
        void *);
```

Parameters

Input

storage The storage to be freed.

Description

pkc_free(3sec) frees any storage that was allocated by any of the **pkc_** routines called by an application. This routine should be used to release contiguous storage returned by any of the **pkc_** routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_free_keyinfo

Purpose

Frees key information storage

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_free_keyinfo(
        keyinfo_t ** keybase);
```

Parameters

Input

keybase Pointer to the **keyinfo_t** structure(s) to be freed.

Description

pkc_free_keyinfo(3sec) frees storage allocated by **pkc_retrieve_keyinfo(3sec)** for a **keyinfo_t** structure.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_free_trustbase

Purpose

Frees a trust base's storage

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_free_trustbase(
        trustbase_t ** base);
```

Parameters

Input

base Specifies trust base whose storage is to be freed.

Description

pkc_free_trustbase(3sec) frees the allocated storage for a trust base.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_free_trustlist

Purpose

Frees a trust list's storage

Synopsis

#include <pkc_api.h>

unsigned32 pkc_free_trustlist(
trustlist_t ** *tr_list***);**

Parameters

Input

tr_list Specifies trust list whose storage is to be freed.

Description

pkc_free_trustlist(3sec) frees the allocated storage for a trust list.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_generic_key_usage.class

Purpose A class that expresses generic key usage

Member Data

Public

- **unsigned long permitted**

Member Functions

Public

- **pkc_generic_key_usage_t()**

  Takes the following argument:

  — **unsigned long permit_bits = 0xffffffffu**

- **char is_permitted() const**

  Takes the following argument:

  — **unsigned long check**

- **char is_permitted() const**

  Takes the following argument:

  — **const pkc_generic_key_usage_t & check**

- **void constrain()**

  Takes the following argument:

  — **unsigned long constraint**

- **void constrain()**

  Takes the following argument:

  — **const pkc_generic_key_usage_t & constraint**

- **void set()**

  Takes the following argument:

  — **unsigned long constraints**

- **pkc_generic_key_usage_t & operator = (unsigned long c)**

Description

pkc_generic_key_usage_t expresses various generic aspects of a key's usage.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_get_key_certifier_count

Purpose

Returns number of key's certifying authorities

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_key_certifier_count(
        keyinfo_t * keyinfobase,
        unsigned key_index,
        size_t * ca_count);
```

Parameters

Input

keyinfobase A **keyinfo_t** structure containing information about the key.

key_index The index of the key (ranging from 0 to *key_count* - 1).

Output

ca_count Number of certifying authorities for the key.

Description

pkc_get_key_certifier_count(3sec) returns the number of certifying authorities in the certification path of the specified key.

The desired information is extracted by the routine from the **keyinfo_t** structure, which must first be obtained by the caller by a call to the **pkc_retrieve_keyinfo(3sec)** routine.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_get_key_certifier_info

Purpose

Returns information about a certifier

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_key_certifier_info(
        keyinfo_t * keyinfobase,
        unsigned key_index,
        unsigned certifier_index,
        char ** certifier_name,
        utc_t * certification_start,
        utc_t * certification_expiration,
        char * crl_valid,
        utc_t * crl_last_seen,
        utc_t * next_crl_expected);
```

Parameters

Input

keyinfobase Information about the key (returned by **pkc_retrieve_keyinfo(3sec)**).

key_index Index of the key.

certifier_index
Index of the certifier about whom information is desired.

Output

certifier_name
The name of the certifier.

certification_start
Time at which certification by this certifier starts.

certification_expiration
Time at which certification by this certifier ends.

crl_valid If TRUE, there is a certificate revocation list for this certifier.

crl_last_seen
Time at which certificate revocation list was last seen.

next_crl_expected
Time at which next certificate revocation list is expected.

Description

pkc_get_key_certifier_info(3sec) returns information about a specific certifier from a key's certification path. Certifier 0 is the CA that vouched for the key; certifier 1 is the CA that vouched for certifier 0, etc. The total number of certifiers for a given key is returned by **pkc_get_key_certifier_count(3sec)**.

The desired information is extracted by the routine from the **keyinfo_t** structure, which must first be obtained by the caller by a call to the **pkc_retrieve_keyinfo(3sec)** routine.

Any of the return parameters may be passed as NULL if the corresponding information is not required.

Upon successful return, the *certifier_name* parameter will contain allocated storage which must be released with **pkc_free(3sec)** when the application has finished with it.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_get_key_count

Purpose

Returns number of keys for a principal

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_key_count(
        keyinfo_t * keyinfobase,
        size_t * key_count);
```

Parameters

Input

keyinfobase Key information returned by **pkc_retrieve_keyinfo(3sec)**.

Output

key_count Number of keys.

Description

pkc_get_key_count(3sec) returns the number of keys within a **keyinfo_t** structure.

The desired information is extracted from the **keyinfo_t** structure, which must first be obtained by the caller by a call to the **pkc_retrieve_keyinfo(3sec)** routine.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_get_key_data

Purpose

Returns a public key

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_key_data(
        keyinfo_t * keyinfobase,
        unsigned key_index,
        unsigned char ** key_data,
        size_t * key_length);
```

Parameters

Input

keyinfobase Key information for the principal, returned by **pkc_retrieve_keyinfo(3sec)**.

key_index Index (ranging from 0 to *key_count* - 1) of the key desired.

Output

key_data The encoded public key.

key_length Length of the key data returned.

Description

pkc_get_key_data(3sec) extracts an encoded public key from a **keyinfo_t** structure. *key_index* is the index of the key (ranging from 0 to *key_count* - 1).

The returned **key_data** is encoded as an ASN.1 BER **SubjectPublicKeyInfo** object (as defined in X.509).

The desired information is extracted by from the **keyinfo_t** structure, which must first be obtained by the caller by a call to the **pkc_retrieve_keyinfo(3sec)** routine.

Upon successful return, *key_data* will contain PKC-allocated storage which must be released with **pkc_free(3sec)** when the application has finished with it.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_get_key_trust_info

Purpose

Returns information about key trust

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_key_trust_info(
        keyinfo_t * keyinfobase,
        unsigned key_index,
        certification_flags_t * flags,
        uuid_t * key_domain,
        unsigned long * key_usages);
```

Parameters

Input

keyinfobase Key information, returned by **pkc_retrieve_keyinfo(3sec)**.

key_index Index of the key, ranging from 0 to **keycount** - 1.

Output

flags Information about the trust that can be placed in the key (see below).

key_domain Indicates domain of retrieved key. A value of **sec_pk_domain_unspecified** or **NULL** means that the policy does not distinguish keys by domain.

key_usages Indicates usage key is intended for.

Description

pkc_get_key_trust_info(3sec) returns a set of flags describing the trust that can be placed in the key.

The desired information is extracted by the routine from the **keyinfo_t** structure, which must first be obtained by the caller by a call to the **pkc_retrieve_keyinfo(3sec)** routine.

The returned **certification_flags_t** structure describes the trust that can be placed in a returned key. It contains the following fields:

- **trust_type**

  A **trust_type_t** value, which will be one of the following:

  — **UNTRUSTED**

  No trust (e.g., unauthenticated).

  — **DIRECT_TRUST**

  Direct trust via third party (e.g., authenticated registry).

  — **CERTIFIED_TRUST**

  Trust certified by caller's trust base.

- **missing_crls**

  A **char**; its value is TRUE (not 0) if one or more CRLs are missing.

- **revoked**

  A **char** whose value is TRUE (not 0) if any certificate has been revoked (even if it was still valid at the retrieval time).

If **key_domain** and **key_usages** are passed as non-**NULL** pointers, upon successful return these parameters will describe the domain and permitted usage(s) of the specified key. Policies that do not distinguish keys according to domain will indicate a domain of **sec_pk_domain_unspecified**; policies that do not distinguish keys according to usage will indicate all usages are permitted.

The returned **key_usages** is a bit mask which describes the usage(s), if any, which the key is restricted to. The value is formed by AND-ing together one or more of the following constants:

PKC_KEY_USAGE_AUTHENTICATION
The key can be used to authenticate a user

PKC_KEY_USAGE_INTEGRITY
The key can be used to provide integrity protection

PKC_KEY_USAGE_KEY_ENCIPHERMENT
The key can be used to encrypt user keys

PKC_KEY_USAGE_DATA_ENCIPHERMENT
The key can be used to encrypt user data

PKC_KEY_USAGE_KEY_AGREEMENT
The key can be used for key-exchange

PKC_KEY_USAGE_NONREPUDIATION
The key can be used for non-repudiation

PKC_CAKEY_USAGE_KEY_CERT_SIGN
The key can be used to sign key certificates

PKC_CAKEY_USAGE_OFFLINE_CRL_SIGN
The key can be used to sign CRLs

PKC_CAKEY_USAGE_TRANSACTION_SIGN
The key can be used to sign transactions

A returned **key_usages** value of **NULL** (or a value with all bits set) means that the key is suitable for any usage.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**,

pkc_get_registered_policies(3sec), **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_get_registered_policies

Purpose

Returns all registered trust policies

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_get_registered_policies(
        gss_OID_set * oid_set);
```

Parameters

Output

oid_set A set of OIDs which represent all installed policies.

Description

pkc_get_registered_policies(3sec) returns a set of OIDs, which point to all currently installed policies (that is, all pre-loaded policies, plus any policies that have been installed via the policy registration API).

An application will call this routine once during its lifetime. After successfully making the call, the application can choose to use the returned OIDs in a call to **pkc_init_trustbase(3sec)**, etc.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_init_trustbase

Purpose

Initializes a trust base

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_init_trustbase(
        trustlist_t ** tr_list,
        gss_OID policy_oid,
        utc_t time,
        selection_t * sel,
        trustbase_t ** base);
```

Parameters

Input

tr_list	Specifies trust list on the basis of which the trust base is to be initialized.
policy_oid	Specifies policy to use.
time	Specifies time at which the public key is to be valid. Can be 0.
sel	Must be set to 0.

Output

base	Initialized trust base.

Description

pkc_init_trustbase(3sec) initializes the initial trust base to include all the certificates initially trusted, given the initial set of trusted certificates. This routine will also store the cross-certificate pair certificates found during the creation of the trust base.

Upon successful return, *base* will contain a PKC-allocated trust base structure, which should be released with **pkc_free_trustbase(3sec)** when the application has finished with it.

Users will normally call the **pkc_** routines in the following order:

1. **pkc_get_registered_policies(3sec)**

   Called once for the lifetime of the application.

2. **pkc_init_trustlist(3sec)**

3. **pkc_append_to_trustlist(3sec)**

   Called one or more times.

   Note that steps 2 and 3 together build up an initial trust list.

4. **pkc_init_trustbase(3sec)**

   A trust base is computed, given an initial trust list.

5. **pkc_retrieve_keylist(3sec)**

   Called one or more times, for each individual's public key that needs to be looked up.

6. **pkc_free_trustlist(3sec)**

7. **pkc_free_trustbase(3sec)**

Return Values

pkc_s_success
: Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_init_trustlist

Purpose

Creates an empty trust list

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_init_trustlist(
        trustlist_t ** tr_list);
```

Parameters

Input

tr_list A PKC-allocated data structure which contains the initialized trust list.

Description

pkc_init_trustlist(3sec) creates an empty trust list. If *tr_list* is empty, returns **pkc_s_asn_bad_param**; if memory cannot be allocated, returns **pkc_s_nomem**; otherwise, returns **pkc_s_success**.

Upon successful return, *tr_list* will contain a PKC-allocated data structure which must be released with **pkc_free_trustlist(3sec)** when the application has finished with it.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_retrieve_keyinfo(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_key_policies.class

Purpose

A class that expresses policy rules and operations

Member Functions

Public

- **pkc_key_policies_t(void)**

  Initializes to ‘‘all policies OK’’.

- **unsigned32 set()**

  Takes the following argument:

  — **const pkc_key_policy_t & pol**

- **unsigned32 set()**

  Adds an allowed policy. Takes the following argument:

  — **const gss_OID pol**

- **unsigned32 set_none(void)**

  Sets ‘‘no policies permitted’’.

- **unsigned32 set_all(void)**

  Sets ‘‘all policies permitted’’.

- **unsigned32 constrain()**

  Takes the following argument:

  — **const pkc_key_policies_t & pol**

- **pkc_key_policies_t & operator = (const pkc_key_policies_t & pol)**

Description

pkc_key_policies_t embodies rules and operations for key policies.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_key_policy.class

Purpose

Key policy class

Member Data

Public

- **gss_OID value**

Member Functions

Public

- **pkc_key_policy_t(void)**
- **virtual ~pkc_key_policy_t()**
- **pkc_key_policy_t & operator = (const pkc_key_policy_t & o)**
- **pkc_key_policy_t & operator = (const gss_OID & o)**

Description

pkc_key_policy_t embodies a key policy.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**,

pkc_revocation.class(3sec), **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_key_usage.class

Purpose

A class that expresses key usage rules

Member Functions

Public

- **pkc_key_usage_t & operator = ()**

  Takes the following argument:

  — **unsigned long c**

- **pkc_key_usage_t()**

  Takes the following argument:

  — **unsigned long c = 0xffffffflu**

Description

pkc_key_usage_t contains usage rules for a key.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Parent Class

This class is derived from **pkc_generic_key_usage_t**.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**,

pkc_name_subtree_constraint.class(3sec),
pkc_name_subtree_constraints.class(3sec), **pkc_pending_revocation.class(3sec)**,
pkc_revocation.class(3sec), **pkc_revocation_list.class(3sec)**,
pkc_trust_list.class(3sec), **pkc_trust_list_element.class(3sec)**,
pkc_trusted_key.class(3sec).

pkc_lookup_element_in_trustlist

Purpose

Retrieves a specified key

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_lookup_element_in_trustlist(
        pkc_trust_list_t * trust_list,
        const pkc_trust_list_element_t ** key,
        unsigned long key_id);
```

Parameters

Input

trust_list Specifies the trust list

key_id Specifies ID of key to return.

Output

key A pointer to the returned key.

Description

pkc_lookup_element_in_trustlist(3sec) takes a trust list and a key id, and returns a pointer to the specified key (if the key actually is in the trust list).

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_copy_trustlist(3sec)**, **pkc_delete_trustlist(3sec)**, **pkc_display_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_lookup_key_in_trustlist

Purpose

Searches a trust list for the specified key

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_lookup_key_in_trustlist(
        pkc_trust_list_t * trust_list,
        const pkc_trusted_key_t ** key,
        unsigned long key_id);
```

Parameters

Input

trust_list Specifies the trust list to search.

key_id Specifies ID of key to return.

Output

key The returned key.

Description

pkc_lookup_key_in_trustlist(3sec) searches the specified trust list for the specified key. In the returned key, the caller will find the following fields.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Fields from the Certificate

- **start_date**

A **utc_t**

- **end_date**

  A **utc_t**

- **ca_usages**

  A **pkc_ca_key_usage_t**

- **user_usages**

  A **pkc_key_usage_t**

- **policies**

  A **pkc_key_policies_t**

- **constraints**

  A **pkc_constraints_t**

Flags:

- **trusted** (A **char**)

  Expresses whether this entry is trusted (*a priori*).

- **certified** (A **char**)

  Expresses whether this key is certified by another entry.

- **certified_by** (**x500name**)

  Name of the CA that certified this key.

- **serial_number** (**asn_integer**)

  Serial number of certifying certificate .

The following fields are copied from the certifying key entry:

- **certified_start_date** (**utc_t**)
- **certified_end_date** (**utc_t**)
- **certified_usages** (**pkc_ca_key_usage_t**)
- **certified_policies** (**pkc_key_policies_t**)
- **certified_constraints** (**pkc_constraints_t**)
- **revoked** (**char**)

Non-zero if the certifying certificate has been revoked.

- **revocation_date** (**utc_t**)

  Date from which certifier revocation is effective.

- *key_id* (**unsigned long**)

  An ID identifying this key entry.

- **ca_key_id** (**unsigned long**)

  The ID of the key that certified this one. 0 means direct trust.

- **old_key_id** (**unsigned long**)

  Temporary storage for use while copying

- **old_ca_key_id** (**unsigned long**)

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**,

pkc_trust_list.class(3sec), **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_lookup_keys_in_trustlist

Purpose

Searches trust list for keys

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_lookup_keys_in_trustlist(
        pkc_trust_list_t * trust_list,
        const pkc_trusted_key_t ** key,
        size_t * key_count,
        const x500name & owner,
        utc_t * key_time ,
        const pkc_generic_key_usage_t * usages );
```

Parameters

Input

trust_list	Specifies trust list to search.
owner	Specifies principal whose keys are to be searched for.
key_time	Specifies time of ownership to search for.
usages	Specifies usage to search for.

Output

key	Array of pointers to keys found.
key_count	Number of keys found.

Description

pkc_lookup_keys_in_trustlist(3sec) searches the specified trust list for keys owned by the specified principal at the specified time for the specified usage. The keys are

returned in an array of pointers to **pkc_trusted_key_t** objects, which is allocated on the heap. The pointers point to elements within the trust list; thus the caller should copy into allocated storage if they are expected to remain valid after the deletion of the trust list.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_name_subord_constraint.class

Purpose Constraint rules and operations class

Member Data

Public

- **pkc_name_subord_constraint_t * next**
- **pkc_name_subord_constraint_t * prev**
- **unsigned long constraint**
- **unsigned skipCerts**

Member Functions

Public

- **void unlink(void)**
- **~pkc_name_subord_constraint_t()**
- **pkc_name_subord_constraint_t()**

  Takes the following argument:

  — **pkc_name_subord_constraints_t * theRoot**
- **pkc_name_subord_constraint_t(void)**
- **void set_constraint()**

  Takes the following argument:

  — **unsigned long c**
- **void set_skipCerts()**

  Takes the following argument:

  — **unsigned c**

- **void get_next_link_constraint() const**

  Generates a new name subordination constraint that will be applicable to a certificate issued by the subject of this constraint. Takes the following argument:

  — **pkc_name_subord_constraint_t ** new_constraint**

- **char is_permitted() const**

  Takes the following arguments:

  — **const x500name & issuer_name**

  — **const x500name & subject_name**

  Return values have following meanings:

1	Permitted
0	Forbidden
–1	Not relevant
–2	Relevant, but explicit permission is required from another subordination constraint.

Description

pkc_name_subord_constraint_t contains name-subordinate constraint rules and operations for a certificate.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**,

pkc_trust_list.class(3sec), **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_name_subord_constraints.class

Purpose A class that expresses subordinate constraints on a name

Member Data

Public

- **pkc_name_subord_constraint_t * first**
- **pkc_name_subord_constraint_t * last**

Member Functions

Public

- **pkc_name_subord_constraints_t &**
  operator = (const pkc_name_subord_constraints_t & o)
- **pkc_name_subord_constraint_t * first**
- **pkc_name_subord_constraint_t * last**
- **pkc_name_subord_constraints_t(void)**
- **~pkc_name_subord_constraints_t()**
- **char is_permitted() const**

  Takes the following arguments:

  — **const x500name & ca_name**

  — **const x500name & subject_name**
- **void get_next_link_constraint() const**

  Takes the following argument:

  — **pkc_name_subord_constraints_t ** new_constraints**

Description

pkc_name_subord_constraints_t embodies a set of subordinate constraints on a name.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_name_subtree_constraint.class

Purpose A class that expresses a subtree constraint on a name

Member Data

Public

- **pkc_name_subtree_constraint_t * next**
- **pkc_name_subtree_constraint_t * prev**
- **x500name * base**
- **x500name * chopBefore**
- **x500name * chopAfter**
- **unsigned minimum**
- **unsigned maximum**

Member Functions

Public

- **void unlink(void)**
- **virtual ~pkc_name_subtree_constraint_t()**
- **pkc_name_subtree_constraint_t()**

  Takes the following argument:

  — **pkc_name_subtree_constraints_t * theRoot**
- **void set_base()**

  Takes the following argument:

  — **const x500name & n**
- **void set_chopBefore()**

Takes the following argument:

— **const x500name & n**

- **void set_chopAfter()**

  Takes the following argument:

  — **const x500name & n**

- **void set_minimum()**

  Takes the following argument:

  — **unsigned n**

- **void set_maximum()**

  Takes the following argument:

  — **unsigned n**

- **char is_permitted() const**

  Takes the following arguments:

  — **const x500name & issuer_name**

  — **const x500name & subject_name**

  Return values have the following meanings:

1	Permitted.
0	Forbidden.
–1	Not relevant.
–2	Relevant, but explicit permission is required from another subtree constraint.

- **void get_next_link_constraint() const**

  Generates a new name subtree constraint that will be applicable to a certificate issued by the subject of this constraint. Takes the following argument:

  — **pkc_name_subtree_constraint_t ** new_constraint**

pkc_name_subtree_constraint.class(3sec)

Description

pkc_name_subtree_constraint_t embodies a subtree constraint on a name.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_name_subtree_constraints.class

Purpose A class that expresses a set of subtree constraints on a name

Member Data

Public

- **pkc_name_subtree_constraint_t * first**
- **pkc_name_subtree_constraint_t * last**

Member Functions

Public

- **pkc_name_subtree_constraints_t &**
  operator = (const pkc_name_subtree_constraints_t & o)
- **pkc_name_subtree_constraint_t * first**
- **pkc_name_subtree_constraint_t * last**
- **pkc_name_subtree_constraints_t(void)**
- **virtual ~pkc_name_subtree_constraints_t()**
- **char is_permitted() const**

  Takes the following arguments:

  — **const x500name & ca_name**

  — **const x500name & subject_name**
- **void get_next_link_constraint() const**

  Takes the following argument:

  — **pkc_name_subtree_constraints_t ** new_constraints**

pkc_name_subtree_constraints.class(3sec)

Description

pkc_name_subtree_constraints_t embodies a set of subtree constraints on a name.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_pending_revocation.class

Purpose

Class of certificates awaiting revocation

Member Data

Public

- **SignedCertificateList crl**

Member Functions

Public

- **pkc_pending_revocation_t()**

  Takes the following arguments:

  — **const SignedCertificateList & crl**

  — **pkc_revocation_list_t * the_root = NULL**

- **pkc_pending_revocation_t &**
  operator = (const pkc_pending_revocation_t & o)
- **virtual void unlink(void)**
- **virtual ~pkc_pending_revocation_t()**

Description

pkc_pending_revocation_t contains certificates awaiting revocation. Has the **friend** class **pkc_revocation_list_t**.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_plcy_delete_keyinfo

Purpose

Frees public key storage

Synopsis

#include <pkc_plcy.h>

unsigned32 pkc_plcy_delete_keyinfo(
gss_OID *policy*,
void ** *keys_handle*);

Parameters

Input

policy Specifies policy module.

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

Description

pkc_plcy_delete_keyinfo(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*delete_keyinfo)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_delete_trustbase

Purpose

Frees trust base storage

Synopsis

#include <pkc_plcy.h>

unsigned32 pkc_plcy_delete_trustbase(
gss_OID *policy***,**
void ** *trust_base_handle***);**

Parameters

Input

policy Specifies policy module.

trust_base_handle
Specifies trust base to be deleted.

Description

pkc_plcy_delete_trustbase(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*delete_trustbase)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_establish_trustbase

Purpose

Establishes a trust base

Name

pkc_plcy_establish_trustbase -

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_establish_trustbase(
        gss_OID policy,
        const pkc_trust_list_t & initial_trust,
        const utc_t * date,
        char initial_explicit_policy_required,
        void ** trust_base_handle);
```

Parameters

Input

policy Specifies policy to use.

initial_trust Specifies the initial set of trusted keys.

date Specifies time for which information is to be returned.

initial_explicit_policy_required
Specifies whether the initial certificate must explicitly contain the active policy in its policies field.

Output

trust_base_handle
The initialized trust base.

pkc_plcy_establish_trustbase(3sec)

Description

pkc_plcy_establish_trustbase(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(establish_trustbase()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

This is a one-time call made by an application to initialize a trust base. It returns an extended trust list. After this call is made, the application can call **pkc_retrieve_keyinfo(3sec)** to obtain the public keys of any particular principal.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_key_certifier_count

Purpose

Returns number of a key's certifying authorities

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_get_key_certifier_count(
        gss_OID policy,
        void * keys_handle,
        unsigned key_index,
        size_t * ca_count);
```

Parameters

Input

policy Specifies policy desired.

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

key_index Specifies the key whose number of certifying authorities is to be returned.

Output

ca_count Number of certifying authorities for the key.

Description

pkc_plcy_get_key_certifier_count(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*get_key_certifier_count)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_key_certifier_info

Purpose

Returns information about a key's certifier

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_get_key_certifier_info(
        gss_OID policy,
        void * keys_handle,
        unsigned key_index,
        unsigned ca_index,
        char ** certifier_name,
        utc_t * certification_start,
        utc_t * certification_expiration,
        char * is_crl_valid,
        utc_t * last_crl_seen,
        utc_t * next_crl_expected);
```

Parameters

Input

policy The policy desired.

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

key_index Index of the key.

ca_index Index of the certifier about whom information is desired.

Output

certifier_name
The name of the certifier.

certification_start
Time at which certification by this certifier starts.

certification_expiration
Time at which certification by this certifier ends.

is_crl_valid If TRUE, there is a certificate revocation list for this certifier.

last_crl_seen
Time at which certificate revocation list was last seen.

next_crl_expected
Time at which next certificate revocation list is expected.

Description

pkc_plcy_get_key_certifier_info(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*get_key_certifier_info)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**,

pkc_plcy_retrieve_key(3sec), **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_key_count

Purpose

Returns number of keys for a principal

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_get_key_count(
        gss_OID policy,
        void * keys_handle,
        size_t * key_count);
```

Parameters

Input

policy Specifies policy desired.

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

Output

key_count The number of principal's keys.

Description

pkc_plcy_get_key_count(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*get_key_count)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_key_data

Purpose

Returns a public key

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_get_key_data(
        gss_OID policy,
        void * keys_handle,
        unsigned key_index,
        unsigned char ** key_data,
        size_t * key_length);
```

Parameters

Input

policy Policy desired.

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

key_index Specifies index of key desired.

Output

key_data The public key requested.

key_length Length of *key_data*.

Description

pkc_plcy_get_key_data(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*get_key_data)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_key_trust

Purpose

Returns information about trust in a key

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_get_key_trust(
        gss_OID policy,
        void * keys_handle,
        unsigned key_index,
        certification_flags_t * flags,
        uuid_t * key_domain,
        unsigned long * key_usages);
```

Parameters

Input

policy Specifies

keys_handle A policy specific structure, obtained from a call to **pkc_plcy_retrieve_keyinfo(3sec)**.

key_index Specifies key about which trust information is requested.

Output

flags Information about the trust that can be placed in the key (see below).

key_domain Indicates domain of retrieved key. A value of **sec_pk_domain_unspecified** or **NULL** means that the policy does not distinguish keys by domain.

key_usages Indicates usage key is intended for.

Description

pkc_plcy_get_key_trust(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(*get_key_data)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

The returned **certification_flags_t** structure describes the trust that can be placed in the key. It contains the following fields:

- **trust_type**

  A **trust_type_t** value, which will be one of the following:

  — **UNTRUSTED**

  No trust (e.g., unauthenticated).

  — **DIRECT_TRUST**

  Direct trust via third party (e.g., authenticated registry).

  — **CERTIFIED_TRUST**

  Trust certified by caller's trust base.

If **key_domain** and **key_usages** are passed as non-**NULL** pointers, upon successful return these parameters will describe the domain and permitted usage(s) of the specified key. Policies that do not distinguish keys according to domain will indicate a domain of **sec_pk_domain_unspecified**; policies that do not distinguish keys according to usage will indicate all usages are permitted.

The returned **key_usages** is a bit mask which describes the usage(s), if any, which the key is restricted to. The value is formed by AND-ing together one or more of the following constants:

PKC_KEY_USAGE_AUTHENTICATION
The key can be used to authenticate a user

PKC_KEY_USAGE_INTEGRITY
The key can be used to provide integrity protection

PKC_KEY_USAGE_KEY_ENCIPHERMENT
The key can be used to encrypt user keys

PKC_KEY_USAGE_DATA_ENCIPHERMENT
The key can be used to encrypt user data

PKC_KEY_USAGE_KEY_AGREEMENT
The key can be used for key-exchange

PKC_KEY_USAGE_NONREPUDIATION
The key can be used for non-repudiation

PKC_CAKEY_USAGE_KEY_CERT_SIGN
The key can be used to sign key certificates

PKC_CAKEY_USAGE_OFFLINE_CRL_SIGN
The key can be used to sign CRLs

PKC_CAKEY_USAGE_TRANSACTION_SIGN
The key can be used to sign transactions

A returned **key_usages** value of **NULL** (or a value with all bits set) means that the key is suitable for any usage.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_get_registered_policies

Purpose

Returns OID set describing registered policies

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_plcy_reg.h>

unsigned32 pkc_plcy_get_registered_policies(
        gss_OID_set * oid_set);
```

Parameters

Output

oid_set A pointer to an OID set describing the currently registered policy implementations.

Description

pkc_plcy_get_registered_policies(3sec) returns an OID set describing the currently registered policy implementations.

pkc_plcy_lookup_policy(3sec) can be called to return details about a specific policy implementation.

Policy modules are identified by OIDs (object identifiers). A policy module is accessed by passing its identifying OID to **pkc_plcy_lookup_policy(3sec)**.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_lookup_policy

Purpose

Returns a policy module descriptor block

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_plcy_reg.h>

unsigned32 pkc_plcy_lookup_policy(
        gss_OID oid,
        pkc_policy_t * details);
```

Parameters

Input

oid An OID identifying a currently registered policy module.

Output

details A pointer to a policy module descriptor block.

Description

pkc_plcy_lookup_policy(3sec) returns a policy module descriptor block for the specified policy, and leaves the policy list unlocked. Calling this routine is the preferred way of obtaining information about a registered policy implementation.

The complete list of registered policies may be obtained by calling **pkc_get_registered_policies(3sec)**.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**, **pkc_register_policy(3sec)**.

pkc_plcy_register_policy

Purpose

Registers a policy module

Synopsis

```
#include <dce/pkc_base.h>
#include <dce/pkc_plcy_reg.h>

unsigned32 pkc_plcy_register_policy(
        pkc_policy_t * plcy,
        int replacement_policy);
```

Parameters

Input

plcy A pointer to the policy module structure to be registered.

replacement_policy
Specifies how the registration is to be handled if an implementation of the policy is already registered. There are three possible values:

PKC_REPLACE_NONE
Specifies that an error should be returned if an implementation of the policy is already registered.

PKC_REPLACE_ENTRYPOINTS
Specifies that only entrypoints that the original implementation (if any) did not provide should be replaced. (Note that this value is not currently supported.)

PKC_REPLACE_ALL
Specifies that the new implementation should replace the existing one, if any.

pkc_plcy_register_policy(3sec)

Description

pkc_plcy_register_policy(3sec) registers a policy module, in the form of a properly declared **pkc_policy_t** data structure, which contains identifying information about the module as well as entry points to all of the module's functions.

Calling this routine will cause the module passed to it to be registered among the system's policy modules; it can then be accessed by other applications via the high level certification routines.

C++ must be used to perform policy registration.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_plcy_retrieve_keyinfo(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**,

pkc_trust_list.class(3sec), **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_plcy_retrieve_keyinfo

Purpose

Retrieves keys for specified principal

Synopsis

```
#include <pkc_plcy.h>

unsigned32 pkc_plcy_retrieve_keyinfo(
        gss_OID policy,
        const void * trust_base_handle,
        const x500name & subjectName,
        const utc_t * date,
        const uuid_t & desired_domain,
        pkc_key_usage_t & desired_usage,
        char initial_explicit_policy_required,
        void ** keys_handle);
```

Parameters

Input

policy Specifies the policy being interrogated.

trust_base_handle
Expresses the caller's initial trust.

subjectName Specifies the desired subject name (principal name).

date Specifies time for which information is to be returned.

desired_domain
Specifies particular domain to which the key-search operation should be restricted. Specify **sec_pk_domain_unspecified** or **NULL** to indicate that keys for any domain should be retrieved.

desired_usage
Allows the user to restrict the key-search operation to keys intended for one or more specific usages.

initial_explicit_policy_required
Specifies whether the initial certificate must explicitly contain the active policy in its policies field.

Output

keys_handle The returned key information.

Description

pkc_plcy_retrieve_keyinfo(3sec) searches the list of registered policies for implementations of the specified policy. If found, the implementation is opened, if necessary, and its **(retrieve_key_info)()** function is invoked. Necessary mutex protection around non-thread safe policy implementations is provided.

The **desired_usage** parameter consists of a bit mask, formed by AND-ing together one or more of the constants:

PKC_KEY_USAGE_AUTHENTICATION
Specifies keys that can be used to authenticate a user

PKC_KEY_USAGE_INTEGRITY
Specifies keys that can be used to provide integrity protection

PKC_KEY_USAGE_KEY_ENCIPHERMENT
Specifies keys that can be used to encrypt user keys

PKC_KEY_USAGE_DATA_ENCIPHERMENT
Specifies keys that can be used to encrypt user data

PKC_KEY_USAGE_KEY_AGREEMENT
Specifies keys that can be used for key-exchange

PKC_KEY_USAGE_NONREPUDIATION
Specifies keys that can be used for non-repudiation

PKC_CAKEY_USAGE_KEY_CERT_SIGN
Specifies keys that can be used to sign key certificates

PKC_CAKEY_USAGE_OFFLINE_CRL_SIGN
Specifies keys that can be used to sign CRLs

PKC_CAKEY_USAGE_TRANSACTION_SIGN
Specifies keys that can be used to sign transactions

A **NULL** can be specified for **desired_usage** to indicate that keys for any usage should be retrieved.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_plcy_intro(3sec)**, **pkc_plcy_delete_keyinfo(3sec)**, **pkc_plcy_delete_trustbase(3sec)**, **pkc_plcy_establish_trustbase(3sec)**, **pkc_plcy_get_key_certifier_count(3sec)**, **pkc_plcy_get_key_certifier_info(3sec)**, **pkc_plcy_get_key_count(3sec)**, **pkc_plcy_get_key_data(3sec)**, **pkc_plcy_get_key_trust(3sec)**, **pkc_plcy_get_registered_policies(3sec)**, **pkc_plcy_lookup_policy(3sec)**, **pkc_plcy_retrieve_key(3sec)**, **pkc_register_policy(3sec)**.

pkc_retrieve_keyinfo

Purpose

Returns information about a key

Synopsis

```
#include <pkc_api.h>

unsigned32 pkc_retrieve_keyinfo(
        trustbase_t * base,
        char * name,
        utc_t * key_date,
        uuid_t * key_domain,
        unsigned long * key_usages,
        selection_t * sel,
        keyinfo_t ** keyinfobase);
```

Parameters

Input

base The trust base, returned by **pkc_init_trustbase(3sec)**.

name Principal name.

key_date Specifies time for which information is to be returned.

key_domain Allows the user to restrict the key-search operation to keys for a particular domain. Specify **sec_pk_domain_unspecified** or **NULL** to indicate that keys for any domain should be retrieved.

key_usages Allows the user to restrict the key-search operation to keys intended for one or more specific usages.

sel Must be 0 (currently ignored).

Output

keyinfobase The returned key information.

Description

pkc_retrieve_keyinfo(3sec) returns a **keyinfo_t** structure describing the set of trusted keys that are valid for the specified principal at the specified date, under any additional constraints specified in *sel*.

The **key_usages** parameter consists of a bit mask, formed by AND-ing together one or more of the constants:

PKC_KEY_USAGE_AUTHENTICATION
The key can be used to authenticate a user

PKC_KEY_USAGE_INTEGRITY
The key can be used to provide integrity protection

PKC_KEY_USAGE_KEY_ENCIPHERMENT
The key can be used to encrypt user keys

PKC_KEY_USAGE_DATA_ENCIPHERMENT
The key can be used to encrypt user data

PKC_KEY_USAGE_KEY_AGREEMENT
The key can be used for key-exchange

PKC_KEY_USAGE_NONREPUDIATION
The key can be used for non-repudiation

PKC_CAKEY_USAGE_KEY_CERT_SIGN
The key can be used to sign key certificates

PKC_CAKEY_USAGE_OFFLINE_CRL_SIGN
The key can be used to sign CRLs

PKC_CAKEY_USAGE_TRANSACTION_SIGN
The key can be used to sign transactions

A **NULL** can be specified for **key_usages** to indicate that keys for any usage should be retrieved.

This routine must be called before any of the following routines can be called:

- **pkc_get_key_count(3sec)**
- **pkc_get_key_data(3sec)**
- **pkc_get_key_trust_info(3sec)**
- **pkc_get_key_certifier_count(3sec)**

- **pkc_get_key_certifier_info(3sec)**

Upon successful return, *keyinfobase* will contain a **keyinfo_t** structure which must be passed in calls to the above routines, which then extract and return the requested information.

The **keyinfo_t** structure must be released by a call to **pkc_free_keyinfo(3sec)** when the application has finished with it.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keylist(3sec)**.

pkc_retrieve_keylist

Purpose

Retrieves all keys for a principal

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_retrieve_keylist(
        trustbase_t * base,
        char * name,
        trusted_key_t ** out_keys,
        size_t * no_of_keys);
```

Parameters

Input

base Specifies trust base from which to retrieve keys.

name Specifies principal whose keys are to be retrieved.

Output

out_keys Keys retrieved.

no_of_keys Number of keys retrieved.

Description

Given an initialized trust base, **pkc_retrieve_keylist(3sec)** returns all public keys for the principal specified.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_intro(3sec)**, **pkc_append_to_trustlist(3sec)**, **pkc_free(3sec)**, **pkc_free_keyinfo(3sec)**, **pkc_free_trustbase(3sec)**, **pkc_free_trustlist(3sec)**, **pkc_get_key_certifier_count(3sec)**, **pkc_get_key_certifier_info(3sec)**, **pkc_get_key_count(3sec)**, **pkc_get_key_data(3sec)**, **pkc_get_key_trust_info(3sec)**, **pkc_get_registered_policies(3sec)**, **pkc_init_trustbase(3sec)**, **pkc_init_trustlist(3sec)**, **pkc_retrieve_keyinfo(3sec)**.

pkc_revocation.class

Purpose A class that expresses certificate revocation operations

Member Data

Public

- **x500name certIssuer**
- **asn_integer certSerialNumber**
- **utc_t certRevocationDate**

Member Functions

Public

- **pkc_revocation_t()**

  Takes the following arguments:

  — **const x500name & issuer**

  — **const asn_integer & serialNumber**

  — **utc_t revocationDate**

  — **pkc_revocation_list_t * the_root = NULL**
- **pkc_revocation_t & operator = (const pkc_revocation_t & o)**
- **virtual void unlink(void)**
- **virtual ~pkc_revocation_t()**

Description

pkc_revocation_t embodies certificate revocation operations. Has **friend** class **pkc_revocation_list_t**.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_revocation_list.class

Purpose Revoked certificates list class

Member Data

Public

- **pkc_revocation_t * first**
- **pkc_revocation_t * last**
- **pkc_pending_revocation_t * first_pending**
- **pkc_pending_revocation_t * last_pending**

Member Functions

Public

- **unsigned32 get_revocation_date() const**

  Takes the following arguments:

  — **const pkc_revocation_t & o**

  — **utc_t * revocationDate**

- **unsigned32 get_revocation_date() const**

  Takes the following arguments:

  — **const x500name & issuer**

  — **const asn_integer & serialNumber**

  — **utc_t * revocationDate**

- **unsigned32 add_revocation()**

  Takes the following argument:

  — **const pkc_revocation_t & o**

- **unsigned32 add_revocation()**

  Takes the following arguments:

  — **const x500name & issuer**

  — **const asn_integer & serialNumber**

  — **const utc_t * revocationDate**

- **unsigned32 add_crl()**

  Takes the following argument:

  — **const SignedCertificateList & crl**

- **unsigned32 add_key()**

  Takes the following arguments:

  — **pkc_trust_list_t * trust_list**

  — **const SubjectPublicKeyInfo & key**

  — **const x500name & subject**

  — **const utc_t & start_date**

  — **const utc_t & end_date**

  — **const pkc_ca_key_usage_t * usages = NULL**

- **pkc_revocation_list_t(void)**
- **virtual ~pkc_revocation_list_t()**
- **void empty(void)**

Description

pkc_revocation_list_t embodies a list of revoked certificates and their dates.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_revoke_certificate

Purpose

Revokes key and dependents from specified trust list

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_revoke_certificate(
        pkc_trust_list_t * trust_list,
        const x500name & issued_by,
        const asn_integer & serial_no,
        utc_t * invalidate_from );
```

Parameters

Input

trust_list Specifies trust list from which to revoke keys.

issued_by Specifies issuer whose keys are to be revoked.

serial_no Specifies serial number of key to revoke.

invalidate_from
Specifies time after which keys will be invalid.

Description

pkc_revoke_certificate(3sec) applies the specified revocation to the specified trust list (i.e. revokes a key and all dependent keys). If *invalidate_from* is NULL, the key is completely revoked; if a valid UTC time is provided, the key is revoked from that time on. The revocation is stored within the trust list, and any subsequent attempts to add the certificate will be rejected.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificates(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_revoke_certificates

Purpose

Revokes a key and all dependent keys

Synopsis

```
#include <pkc_certs.h>

unsigned32 pkc_revoke_certificates(
        pkc_trust_list_t * trust_list,
        const SignedCertificateList * crl);
```

Parameters

Input

trust_list Specifies list from which keys are to be revoked.

crl Specifies keys to revoke.

Description

pkc_revoke_certificates(3sec) applies the specified revocations to the specified trust list (i.e. revokes a key and all dependent keys). The revocations are stored within the trust list, and any subsequent attempts to add a revoked certificate will be rejected.

This routine is a C++ interface. C++ must be used to perform direct certificate manipulation.

See also the contents of the **asn.h** and **x509.h** header files, which define some of the basic types used by the low-level certificate manipulation routines.

Return Values

pkc_s_success
Operation successfully completed.

Errors

Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Related Information

Functions: **pkc_add_trusted_key(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**. Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_trust_list.class

Purpose A class that expresses certificate trust list operations

Member Data

Public

- **pkc_trust_list_element_t * first**
- **pkc_trust_list_element_t * last**
- **pkc_revocation_list_t revocation_list**

  List of revocations

Member Functions

Public

- **pkc_trust_list_t(void)**
- **void empty(void)**
- **virtual ~pkc_trust_list_t()**
- **pkc_trust_list_t & operator = (const pkc_trust_list_t & o)**
- **unsigned32 fixup_links(void)**
- **unsigned32 find_certified_key()**

  Returns the first key entry that was created from the specified certificate. Call **find_next_certified_key()** to return the next such key. Takes the following arguments:

  — **const x500name &** *certifier*

  — **const asn_integer &** *certifying_serial_no*

  — **pkc_trust_list_element_t **** *key*
- **unsigned32 find_next_certified_key()**

Returns the next key entry that was created from the same certificate as the current entry. *key* is both an input and an output. Takes the following argument:

— **pkc_trust_list_element_t **** *key*

- **unsigned32 find_certified_key_by_id()**

  Returns the first key entry that was certified by the specified key id. Call **find_next_certified_key_by_id()** to return the next such key. Takes the following arguments:

  — **unsigned long** *ca_key_id*

  — **pkc_trust_list_element_t **** *key*

- **unsigned32 find_next_certified_key_by_id()**

  Returns the next key entry that was certified by the same key as the current entry. *key* is both an input and an output. Takes the following argument:

  — **pkc_trust_list_element_t **** *key*

Description

pkc_trust_list_t embodies rules and operations for a certificate trust list. This class has the **friend** class **pkc_trust_list_element_t**.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_trust_list_element.class

Purpose Public key class

Member Data

Public

- **pkc_trust_list_element_t * next**
- **pkc_trust_list_element_t * prev**

Member Functions

Public

- **void unlink(void)**
- **pkc_trust_list_element_t()**

  Takes the following argument:

  — **pkc_trust_list_t &** *the_root*
- **pkc_trust_list_element_t()**

  Takes the following arguments:

  — **pkc_trust_list_t &** *the_root*

  — **utc_t** *startDate*

  — **utc_t** *endDate*

  — **pkc_ca_key_usage_t** *caUsages*

  — **pkc_key_usage_t** *userUsages*

  — **pkc_key_policies_t** *keyPolicies*

  — **pkc_constraints_t** *keyConstraints*
- **virtual ~pkc_trust_list_element_t()**

- **unsigned32 apply_revocation()**

  Apply a revocation to this key, starting at the specified date. If revocation_date is NULL, the key is completely revoked: this key, and all keys dependent on it will be revoked. Takes the following argument:

  — **utc_t *** *revocation_date*

Description

pkc_trust_list_element_t defines a key.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Parent Class

This class is derived from the **pkc_trusted_key_t** class. It has as **friend** class **pkc_trust_list_t** (a list of trusted keys).

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trusted_key.class(3sec)**.

pkc_trusted_key.class

Purpose Trusted public key class

Member Data

Public

- **SubjectPublicKeyInfo value**
- **x500name owner**

Member Functions

Public

- **pkc_trusted_key_t(void)**
- **pkc_trusted_key_t()**

  Takes following arguments:

  — **utc_t** *startDate*

  — **utc_t** *endDate*

  — **pkc_ca_key_usage_t** *caUsages*

  — **pkc_key_usage_t** *userUsages*

  — **pkc_key_policies_t** *keyPolicies*

  — **pkc_constraints_t** *keyConstraints*
- **unsigned32 get_start_date() const**

  Takes the following argument:

  — **utc_t *** *start_date*
- **unsigned32 get_end_date() const**

  Takes the following argument:

— **utc_t *** *start_date*

- **unsigned32 get_usages() const**

  Takes the following argument:

  — **pkc_key_usage_t *** *user_usages*

- **unsigned32 get_ca_usages() const**

  Takes the following argument:

  — **pkc_ca_key_usage_t *** *ca_usages*

- **unsigned32 get_key_policies() const**

  Takes the following argument:

  — **pkc_key_policies_t *** *policies*

- **unsigned32 get_constraints() const**

  Takes the following argument:

  — **pkc_constraints_t *** *constraints*

- **unsigned32 get_certifier() const**

  Takes the following argument:

  — **x500name &** *name*

- **unsigned32 get_certifier() const**

  Takes the following argument:

  — **pkc_trusted_key_t **** *ca*

- **char valid_at() const**

  Takes the following argument:

  — **utc_t *** *time*

- **pkc_trusted_key_t & operator = (const pkc_trusted_key_t & o)**

- **char may_certify() const**

  Takes the following arguments:

  — **const x500name &** *subject*

  — **unsigned long** *usage* **= PKC_CAKEY_USAGE_KEY_CERT_SIGN**

- **char may_certify() const**

  Takes the following arguments:

  — **const x500name & subject**

  — **const pkc_ca_key_usage_t & usage**

Description

pkc_trusted_key_t is a class that expresses trust in a public key. It is very much like a certificate, but with trust pre-established, rather than based on a signature.

This class has the **friend** class **pkc_trust_list_t**.

The certificate manipulation routines are a C++ interface. C++ must be used to perform direct certificate manipulation.

Related Information

Classes: **pkc_ca_key_usage.class(3sec)**, **pkc_constraints.class(3sec)**, **pkc_generic_key_usage.class(3sec)**, **pkc_key_policies.class(3sec)**, **pkc_key_policy.class(3sec)**, **pkc_key_usage.class(3sec)**, **pkc_name_subord_constraint.class(3sec)**, **pkc_name_subord_constraints.class(3sec)**, **pkc_name_subtree_constraint.class(3sec)**, **pkc_name_subtree_constraints.class(3sec)**, **pkc_pending_revocation.class(3sec)**, **pkc_revocation.class(3sec)**, **pkc_revocation_list.class(3sec)**, **pkc_trust_list.class(3sec)**, **pkc_trust_list_element.class(3sec)**.

Functions: **pkc_add_trusted_key(3sec)**, **pkc_lookup_keys_in_trustlist(3sec)**, **pkc_lookup_key_in_trustlist(3sec)**, **pkc_check_cert_against_trustlist(3sec)**, **pkc_revoke_certificate(3sec)**, **pkc_revoke_certificates(3sec)**, **pkc_delete_trustlist(3sec)**, **pkc_copy_trustlist(3sec)**.

rdacl_get_access

Purpose

Reads a privilege attribute certificate

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_get_access(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_acl_permset_t *net_rights,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the object whose ACL is to be accessed.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

Output

net_rights The output list of access rights, in **sec_acl_permset_t** form. This is a 32-bit set of permission flags supported by the manager type.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_get_access()** routine determines the complete extent of access to the specified object by the calling process. Although the **rdacl_test_access()** routines are the preferred method of testing access, this routine is useful for implementing operations like the conventional UNIX access function.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
: The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_invalid_manager_type
: The manager type is not valid.

sec_acl_invalid_acl_type
: The ACL type is not valid.

sec_acl_not_authorized
: The requested operation is not allowed.

sec_acl_object_not_found
: The requested object could not be found.

error_status_ok
The call was successful.

Related Information

Functions: **rdacl_test_access(3sec)**, **sec_intro(3sec)**.

rdacl_get_manager_types

Purpose

Lists the types of ACLs protecting an object

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_get_manager_types(
        handle_t h,
        sec_acl_component_name_t component_name,
        sec_acl_type_t sec_acl_type,
        unsigned32 size_avail,
        unsigned32 *size_used,
        unsigned32 *num_types,
        uuid_t manager_types[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object.

component_name
A character string containing the name of the target object.

sec_acl_type The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

size_avail An unsigned 32-bit integer containing the allocated length of the *manager_types[ ]* array.

Output

size_used An unsigned 32-bit integer containing the number of output entries returned in the *manager_types[ ]* array.

num_types An unsigned 32-bit integer containing the number of types returned in the *manager_types[ ]* array. This is always equal to *size_used*.

manager_types[ ]
An array of length *size_avail* to contain UUIDs (of type **uuid_t**) identifying the different types of ACL managers protecting the target object.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_get_manager_types()** routine returns a list of the types of ACLs protecting an object. For example, in addition to the regular file system ACL, a file representing the stable storage of some database could have an ACL manager that supported permissions allowing database updates only on certain days of the week.

ACL editors and browsers can use this operation to determine the ACL manager types that a particular reference monitor is using to protect a selected entity. Then, using the **rdacl_get_printstring()** routine, they can determine how to format for display the permissions supported by a specific manager.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **rdacl_get_printstring(3sec)**, **sec_intro(3sec)**.

rdacl_get_mgr_types_semantics

Purpose

Lists the ACL manager types protecting an object and the POSIX semantics supported by each manager type

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_get_mgr_types_semantics(
        handle_t h,
        sec_acl_component_name_t component_name,
        sec_acl_type_t sec_acl_type,
        unsigned32 size_avail,
        unsigned32 *size_used,
        unsigned32 *num_types,
        uuid_t manager_types[ ],
        sec_acl_posix_semantics_t posix_semantics[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object.

component_name
A character string containing the name of the target object.

sec_acl_type The ACL type used to limit the function's output to ACL managers that control the specified types of ACLs. The possible values are as follows:

- **sec_acl_type_object**

  Object ACL, the ACL controlling access to an object.

- **sec_acl_type_default_object**

Initial Object ACL, the default ACL for objects created in a container object.

- **sec_acl_type_default_container**

  Initial Container ACL, the default ACL for containers created in a container object.

size_avail An unsigned 32-bit integer containing the allocated length of the *manager_types[ ]* and the *posix_semantics[ ]* arrays.

Output

size_used An unsigned 32-bit integer containing the number of output entries returned in the *manager_types[ ]* array.

num_types An unsigned 32-bit integer containing the number of types returned in the *manager_types[ ]* array. This is always equal to *size_used*.

manager_types[ ]
An array of length *size_avail* containing the returned UUIDs (of type **uuid_t**) identifying the different ACL manager types protecting the target object.

posix_semantics[ ]
An array of length *size_avail* containing the POSIX semantics (of type **sec_acl_posix_semantics_t**) that are supported by each returned ACL manager type.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_get_manager_types_semantics()** routine returns a list of the ACL manager types protecting an object and a list of the POSIX semantics supported by those ACL manager types. Access to an object can be controlled by multiple ACL manager types. For example, access to a file representing the stable storage of a database could be controlled by two ACL manager types each with completely different sets of permissions: one to provide standard file system access (read, write, execute, and so on) and one to provide access that allows database updates only on certain days of the week.

ACL editors and browsers can use this operation to determine the ACL manager types that a particular reference monitor is using to protect a selected entity. Then, using the **rdacl_get_printstring()** routine, they can determine how to format for display the permissions supported by a specific manager.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
: The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
: The call was successful.

Related Information

Functions: **rdacl_get_printstring(3sec)**, **sec_intro(3sec)**.

rdacl_get_printstring

Purpose

Returns printable ACL strings

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_get_printstring(
        handle_t h,
        uuid_t *manager_type,
        unsigned32 size_avail,
        uuid_t *manager_type_chain,
        sec_acl_printstring_t *manager_info,
        boolean32 *tokenize,
        unsigned32 *total_num_printstrings,
        unsigned32 *size_used,
        sec_acl_printstring_t printstrings[],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **rdacl_get_manager_types()** to acquire a list of the manager types protecting a given object.

size_avail An unsigned 32-bit integer containing the allocated length of the *printstrings*[ ] array.

Output

manager_type_chain
: If the target object ACL contains more than 32 permission bits, multiple manager types are used, one for each 32-bit wide slice of permissions. The UUID returned in *manager_type_chain* refers to the next ACL manager in the chain. If there are no more ACL managers for this ACL, **uuid_nil** is returned.

manager_info
: Provides a name and helpstring for the given ACL manager.

tokenize
: When FALSE this variable indicates that the returned permission printstrings are unambiguous and therefore may be concatenated when printed without confusion. When TRUE, however, this property does not hold, and the strings need to be separated when printed or passed.

total_num_printstrings
: An unsigned 32-bit integer containing the total number of permission printstrings supported by this ACL manager type.

size_used
: An unsigned 32-bit integer containing the number of permission entries returned in the *printstrings*[ ] array.

printstrings[ ]
: An array of permission printstrings of type **sec_acl_printstring_t**. Each entry of the array is a structure containing three components:

  printstring
  : A character string of maximum length **sec_acl_printstring_len** containing the printable representation of a specified permission.

  helpstring
  : A character string of maximum length **sec_acl_printstring_help_len** containing some text that can be used to describe the specified permission.

  permissions
  : A **sec_acl_permset_t** permission set describing the permissions that are to be represented with the companion printstring.

    The array consists of one such entry for each permission supported by the ACL manager identified by *manager_type*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_get_printstring()** routine returns an array of printable representations (called printstrings) for each permission bit or combination of permission bits the specified ACL manager will support. The ACL manager type specified must be one of the types indicated by the ACL handle.

In addition to returning the printstrings, this routine also returns instructions about how to print the strings. When the *tokenize* variable is set to FALSE, a print string might be **r** or **w**, which could be concatenated in the display as **rw** without any confusion. However, when the *tokenize* variable is TRUE, it implies the printstrings might be of a form like **read** or **write**, which must be displayed separated by spaces or colons or something.

In any list of permission printstrings, there may appear to be some redundancy. ACL managers often define aliases for common permission combinations. By convention, however, simple entries need to appear at the beginning of the *printstrings*[ ] array, and combinations need to appear at the end.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not among those referenced by the input handle.

error_status_ok
The call was successful.

Related Information

Functions: **rdacl_get_manager_types(3sec)**, **sec_acl_bind(3sec)**, **sec_intro(3sec)**.

rdacl_get_referral

Purpose

Gets a referral to an ACL update site

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_get_referral(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_acl_type_t sec_acl_type,
        sec_acl_tower_set_t *towers[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

sec_acl_type The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**

- **sec_acl_type_default_container**

Output

towers[ ] A pointer to address information indicating an ACL update site. This information, obtained from the RPC runtime, is used by the client-side code to construct a new ACL binding handle indicating a site that will not return the **sec_acl_site_readonly** error.

The **sec_acl_tower_set_t** structure contains an array of towers (called *towers[ ]*) and an unsigned 32-bit integer indicating the number of array elements (called *count*). This type enables the client to pass in an unallocated array of towers and have the server allocate the correct amount.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_get_referral()** routine obtains a referral to an ACL update site. This function is used when the current ACL site yields a **sec_acl_site_readonly** error. Some replication managers will require all updates for a given object to be directed to a given replica. If clients of the generic ACL interface know they are dealing with an object that is replicated in this way, this function allows them to recover from the problem and rebind to the proper update site. The DCE network registry, for example, is replicated this way.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

rdacl_lookup

Purpose

Returns the ACL for an object

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_lookup(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_acl_type_t sec_acl_type,
        sec_acl_result_t *result);
```

Parameters

Input

h
A handle referring to the target object.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

sec_acl_type
The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**

- **sec_acl_type_default_container**

Output

result A pointer to a tagged union of type **sec_acl_result_t**. The tag is the completion status, **result.st**. If **result.st** is equal to **error_status_ok**, the union contains an ACL. Otherwise, the completion status indicates an error, and the union is empty.

If the call returned successfully, the **result.tagged_union.sec_acl_list_t** structure contains a **sec_acl_list_t**. This data type is an array of pointers to **sec_acl_t**s that define ACLs. If the permission set of the returned ACL is 32 bits or smaller, **sec_acl_list_t** points to only one **sec_acl_t**. If the permission set of the returned ACL is larger than 32 bits, multiple **sec_acl_t**s are used to hold them, and the **sec_acl_list_t** points to multiple **sec_acl_t**s.

Description

The **rdacl_lookup()** routine loads into memory a copy of an object's ACL corresponding to the specified manager type. The routine returns a pointer to the ACL. This routine is only used by ACL editors and browsers; an application would use **sec_acl_test_access()** or **sec_acl_test_access_on_behalf()** to process the contents of an ACL.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

sec_acl_cant_allocate_memory
The requested operation requires more memory than is available.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_test_access(3sec)**, **sec_acl_test_access_on_behalf(3sec)**, **sec_intro(3sec)**.

rdacl_replace

Purpose

Replaces an ACL

Synopsis

```
#include <dce/rdaclif.h>

void rdacl_replace(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_acl_type_t sec_acl_type,
        sec_acl_list_t *sec_acl_list,
        error_status_t *status);
```

Parameters

Input

h
A handle referring to the target object.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

sec_acl_type
The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**

- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

sec_acl_list

The new ACL to use for the target object. This is represented by a pointer to the **sec_acl_list_t** structure containing the complete access control list. An ACL contains a list of ACL entries, the UUID of the default cell where authentication takes place (foreign entries in the ACL contain the name of their parent cell), and the UUID of the ACL manager to interpret the list.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_replace()** routine replaces the ACL indicated by the input handle with the information in the *sec_acl_list* parameter. ACLs are thought of as immutable, and in order to modify them, an editing application must read an entire ACL (using the **sec_acl_lookup()** routine), modify it as needed, and replace it using this routine.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl

The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_lookup(3sec)**, **sec_intro(3sec)**.

rdacl_test_access

Purpose

Tests access to an object

Synopsis

```
#include <dce/rdaclif.h>

boolean32 rdacl_test_access(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_acl_permset_t desired_permset,
        error_status_t *status);
```

Parameters

Input

h
A handle referring to the target object.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

desired_permset
A permission set in **sec_acl_permset_t** form containing the desired privileges. This is a 32-bit set of permission flags supported by the manager type.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_test_access()** routine determines if the specified ACL contains entries granting privileges to the calling process matching those in *desired_permset*. An application generally only inquires after the minimum set of privileges needed to accomplish a specific task.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **rdacl_test_access_on_behalf(3sec)**, **sec_intro(3sec)**.

rdacl_test_access_on_behalf

Purpose

Tests access to an object on behalf of another process

Synopsis

```
#include <dce/rdaclif.h>

boolean rdacl_test_access_on_behalf(
        handle_t h,
        sec_acl_component_name_t component_name,
        uuid_t *manager_type,
        sec_id_pac_t *subject,
        sec_acl_permset_t desired_permset,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object.

component_name
A character string containing the name of the target object.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

subject A privilege attribute certificate (PAC) for the subject process. The PAC contains the name and UUID of the principal and parent cell of the subject process, as well as a list of any groups to which it belongs. The PAC also contains a flag (named **authenticated**). When set, it indicates that the certificate was obtained from an authenticated source. When not set, the certificate must not be trusted.

The field is FALSE when it was obtained from the **rpc_auth** layer and the protect level was set to **rpc_c_protect_level_none**. This indicates that no authentication protocol was actually used in the remote procedure call; the identity was simply transmitted from the caller to the callee. If an authentication protocol was used, then the flag is set to TRUE. A server uses **rpc_binding_inq_auth_client()** to acquire a certificate for the client process.

desired_permset
A permission set in **sec_acl_permset_t** form containing the desired privileges. This is a 32-bit set of permission flags supported by the manager type.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rdacl_test_access_on_behalf()** routine determines if the specified ACL contains entries granting privileges to the subject, a process besides the calling process, matching those in *desired_permset*. This routine succeeds only if the access is available to both the caller process as well as the subject identified in the call. An application will generally only inquire after the minimum set of privileges needed to accomplish a specific task.

Notes

This call is not intended to be used by application programs. The *sec_acl* application programming interface (API) provides all the functionality necessary to use the ACL facility. This reference page is provided for programmers who wish to write an ACL manager. In order to write an ACL manager, a programmer must implement the entire **rdacl** interface.

This network interface is called on the client side via the *sec_acl* local interface. Developers are responsible for implementing the server side of this interface. Test server code is included as a sample implementation.

Files

/usr/include/dce/rdaclif.idl
The **idl** file from which **dce/rdaclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **rdacl_test_access(3sec)**, **rpc_binding_inq_auth_client(3rpc)**, **sec_intro(3sec)**.

rsec_pwd_mgmt_gen_pwd

Purpose

Generates a set of passwords

Synopsis

```
#include <dce/rsec_pwd_mgmt.h>

void rsec_pwd_mgmt_gen_pwd(
        handle_t pwd_mgmt_svr_h,
        sec_rgy_name_t princ_name,
        unsigned32 plcy_args,
        sec_attr_t plcy[ ],
        sec_bytes_t gen_info_in,
        unsigned32 num_pwds,
        unsigned32 *num_returned,
        sec_passwd_rec_t gen_pwd_set[ ],
        sec_bytes_t *gen_info_out,
        error_status_t *stp);
```

Parameters

Input

pwd_mgmt_svr_h
: An RPC binding handle to the password management server exporting this operation.

princ_name
: The name of the principal requesting the generated passwords.

plcy_args
: The size of the *plcy[ ]* array.

plcy[ ]
: An array of extended registry attributes, each specifying a password management policy of some sort. The contents of this array are as follows:

  plcy[0] Effective registry password minimum length for the principal.

plcy[1] Effective registry password policy flags for the principal, describing limitations on password characters.

gen_info_in An NDR pickle containing additional information needed to generate the passwords. There are currently no encoding types defined.

num_pwds The number of generated passwords requested.

Output

num_returned
The number of generated passwords returned.

gen_pwd_set[ ]
An array of generated passwords, each stored in a **sec_passwd_rec_t** structure.

gen_info_out An NDR pickle containing additional information returned by the password management server. There are currently no encoding types defined.

stp A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rsec_pwd_mgmt_gen_pwd()** routine returns a set of generated passwords.

Notes

This function is not intended to be called by application programmers. The **sec_pwd_mgmt()** API provides all the functionality necessary to retrieve generated passwords. This reference page is provided for programmers who want to write their own password management servers.

This network interface is called on the client side via the **sec_pwd_mgmt_gen_pwd()** operation. Developers are responsible for implementing the server side of this interface. (**pwd_strengthd(8sec)** is provided as a sample implementation.)

The **plcy**[ ] parameter is intended to be expandable to allow administrators to attach new password policy ERAs to a principal. This feature is, however, currently unsupported, and the **plcy**[ ] parameter consists only of the entries described in this reference page.

Files

/usr/include/dce/sec_pwd_mgmt.idl
The **idl** file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_pwd_mgmt_not_authorized
The user is not authorized to call this API.

sec_pwd_mgmt_svr_error
Password management server generic error. Additional information is usually logged by the password management server.

error_status_ok
The call was successful.

Related Information

Functions: **pwd_strengthd(8sec)**, **rsec_pwd_mgmt_str_chk(3sec)**, **sec_intro(3sec)**, **sec_pwd_mgmt_gen_pwd(3sec)**.

rsec_pwd_mgmt_str_chk

Purpose

Strength-checks a password

Synopsis

```
#include <dce/rsec_pwd_mgmt.h>

boolean32 rsec_pwd_mgmt_str_chk(
        handle_t handle,
        sec_rgy_name_t princ,
        sec_passwd_rec_t *pwd,
        signed32 pwd_val_type,
        unsigned32 plcy_args,
        sec_attr_t plcy[ ],
        sec_bytes_t str_info_in,
        sec_bytes_t *str_info_out,
        error_status_t *stp);
```

Parameters

Input

handle An RPC binding handle to the password management server exporting this operation.

princ The name of the principal requesting the generated passwords.

pwd A pointer to the password to be strength checked.

pwd_val_type
The value of the user's password validation type (as stored in the *pwd_val_type* ERA).

plcy_args The size of the *plcy[ ]* array.

plcy[ ] An array of extended registry attributes, each specifying a password management policy of some sort. The contents of this array are as follows:

plcy[0] Effective registry password minimum length for the principal.

plcy[1] Effective registry password policy flags for the principal, describing limitations on password characters.

str_info_in An NDR pickle containing additional information needed to strength check the password. There are currently no encoding types defined.

Output

str_info_out An NDR pickle containing additional information returned by the password management server. There are currently no encoding types defined.

stp A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **rsec_pwd_mgmt_str_chk()** routine strength checks a password.

Notes

This function is not intended to be called by application programmers. The registry server provides all the functionality necessary to strength check passwords. This reference page is provided for programmers who wish to write their own password management servers.

This network interface is called on the client side via **secd(8)**. Developers are responsible for implementing the server side of this interface. (**pwd_strengthd(8sec)** is provided as a sample implementation.)

The **plcy**[ ] parameter is intended to be expandable to allow administrators to attach new password policy ERAs to a principal. This feature is, however, currently unsupported, and the **plcy**[ ] parameter consists only of the entries described in this reference page.

Return Value

The **rsec_pwd_mgmt_str_chk()** routine returns TRUE if the user's password passes the server's strength checking algorithm and FALSE if it does not.

Files

/usr/include/dce/sec_pwd_mgmt.idl
The **idl** file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_pwd_mgmt_str_check_failed
The password failed the server's strength checking algorithm.

sec_pwd_mgmt_not_authorized
The user is not authorized to call this API.

sec_pwd_mgmt_svr_error
Password management server generic error. Additional information is usually logged by the password management server.

error_status_ok
The call was successful

Related Information

Functions: **pwd_strengthd(8sec)**, **rsec_pwd_mgmt_gen_pwd(3sec)**, **sec_intro(3sec)**.

sec_acl_bind

Purpose

Returns a handle for an object's ACL

Synopsis

```
#include <dce/daclif.h>

void sec_acl_bind(
        unsigned char *entry_name,
        boolean32 bind_to_entry,
        sec_acl_handle_t *h,
        error_status_t *status);
```

Parameters

Input

entry_name The name of the target object. Subsequent ACL operations using the returned handle will affect the ACL of this object.

bind_to_entry
Bind indicator, for use when *entry_name* identifies both an entry in the global namespace and an actual object. A TRUE value binds the handle to the entry in the namespace, while FALSE binds the handle to the actual object.

Output

h A pointer to the **sec_acl_handle_t** variable to receive the returned ACL handle. The other *sec_acl* routines use this handle to refer to the ACL for the object specified with *entry_name*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_bind()** routine returns a handle bound to the indicated object's ACL. This routine is central to all the other *sec_acl* routines, each of which requires this handle to identify the ACL on which to operate.

Notes

If the specified name is both an actual object, and an entry in the global namespace, there are two ACLs associated with it. For example, in addition to the ACL normally attached to file system objects, the root directory of a file system has an ACL corresponding to its entry in the global namespace. This controls access by outsiders to the entire file system, whereas the resident ACL for the root directory only controls access to the directory and, by inheritance, its subdirectories. The ambiguity must be resolved with the *bind_to_entry* parameter.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_object_not_found
The requested object could not be found.

sec_acl_no_acl_found
There is no ACL associated with the specified object.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_acl_bind_auth

Purpose

Returns an opaque handle to an object's ACL

Synopsis

```
#include <dce/daclif.h>

void sec_acl_bind(
        unsigned char *entry_name,
        boolean32 bind_to_entry,
        sec_acl_bind_auth_info_t *auth_info,
        sec_acl_handle_t *h,
        error_status_t *status);
```

Parameters

Input

entry_name The name of the target object. Subsequent access control list (ACL) operations using the returned handle will affect the ACL of this object.

bind_to_entry
A bind indicator, for use when *entry_name* identifies both an entry in the global namespace and an actual object. A TRUE value binds the handle to the entry in the namespace, while FALSE binds the handle to the actual object.

auth_info A pointer to the **sec_acl_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info(3rpc)** reference page for more information on authorization.) If this argument is not supplied, default authorization information is used as it is in the **sec_acl_bind()** routine.

Output

h A pointer to the **sec_acl_handle_t** variable to receive the returned ACL handle. The other *sec_acl* routines use this handle to refer to the ACL for the object specified with *entry_name*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_bind_auth()** routine returns a handle bound to the indicated object's ACL. This routine and the **sec_acl_bind()** routine provide the handle that identifies the ACL on which other *sec_acl* routines operate. Use this routine instead of the **sec_acl_bind()** routine to specify authorization information explicitly instead of using the default authorization information.

Note: If the specified name is both an actual object, and an entry in the global namespace, there are two ACLs associated with it. For example, in addition to the ACL normally attached to file system objects, the root directory of a file system has an ACL corresponding to its entry in the global namespace. This controls access by outsiders to the entire file system, whereas the resident ACL for the root directory only controls access to the directory and, by inheritance, its subdirectories. The ambiguity must be resolved with the *bind_to_entry* parameter.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_object_not_found
The requested object could not be found.

sec_acl_no_acl_found
There is no ACL associated with the specified object.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_acl_bind(3sec)**.

sec_acl_bind_to_addr

Purpose

Returns a handle to an object identified by its network address

Synopsis

```
#include <dce/daclif.h>

void sec_acl_bind_to_addr(
        unsigned char *site_addr,
        sec_acl_component_name_t component_name,
        sec_acl_handle_t *h,
        error_status_t *status);
```

Parameters

Input

site_addr An RPC string binding to the fully qualified network address of the target object.

component_name
The name of the target object. Subsequent ACL operations using the returned handle will affect the ACL of this object.

Output

h A pointer to the **sec_acl_handle_t** variable to receive the returned ACL handle. The other *sec_acl* routines use this handle to refer to the ACL for the object specified with *entry_name*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_bind_to_addr()** routine returns a handle bound to the indicated object's ACL manager. This routine and the **sec_acl_bind()** routine are central to all the other *sec_acl* routines, each of which requires a handle to identify the ACL on which to operate.

This routine differs from **sec_acl_bind()** in that it binds to the network address of the target object, rather than to a cell namespace entry. Therefore, unlike **sec_acl_bind()**, it is possible to pass **sec_acl_bind_to_addr()** a null string as a component name and to bind with a nonexistent name. The purpose of this call is to eliminate the necessity of looking up an object's name. To validate the name, use **sec_acl_bind()**.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_object_not_found
The requested object could not be found.

sec_acl_no_acl_found
There is no ACL associated with the specified object.

sec_acl_unable_to_authenticate
The call could not authenticate to the server that manages the target object's ACL.

sec_acl_bind_error
The call could not bind to the requested site.

sec_acl_invalid_site_name
The *site_addr* parameter is invalid.

sec_acl_cant_allocate_memory
Memory allocation failure.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_acl_calc_mask

Purpose

Returns the **sec_acl_type_mask_obj** entry for the specified ACL list

Synopsis

```
#include <dce/daclif.h>

void sec_acl_calc_mask(
        sec_acl_list_t *sec_acl_list,
        error_status_t *status);
```

Parameters

Input/Output

sec_acl_list

A pointer to a **sec_acl_type_t** the specifies the number of ACLs of each ACL type. The **sec_acl_type_t** data type distinguishes between the various types of ACLs an object can possess for a given manager. In the file system, for example, most objects have only one ACL, controlling the access to that object, but objects that control the creation of other objects (sometimes referred to as *containers*) may have more. A directory, for example, can have ACLs to be used as initial values when member objects are created.

Do not confuse ACL types with the permissions corresponding to different ACL manager types or with the ACL manager types themselves.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_calc_mask()** routine calculates and sets the **sec_acl_e_type_mask_obj** entry of the specified ACL list. The value of the **sec_acl_e_type_mask_obj** entry is the union of the permissions of all ACL entries that refer to members of the file group class.

This operation is performed locally, within the client. The function does not check to determine if the manager to which the specified ACL list will be submitted supports the **sec_acl_e_type_mask_obj** entry type. The calling application must determine whether to call this routine, after obtaining the required, if any, POSIX semantics, via the **sec_acl_get_mgr_types_semantics()** routine.

Notes

This call is provided in source code form.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_cant_allocate_memory
Requested operation requires more memory than is available.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_acl_get_access

Purpose

Lists the access (permission set) that the caller has for an object

Synopsis

```
#include <dce/daclif.h>

void sec_acl_get_access(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        sec_acl_permset_t *net_rights,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the object whose ACL is to be accessed. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the manager type of the ACL in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

Output

net_rights
The output list of access rights in **sec_acl_permset_t** form. This is a 32-bit set of permission flags supported by the manager type.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_get_access()** routine determines the complete extent of access to the specified object by the calling process. Although the **sec_acl_test_access()** and **sec_acl_test_access_on_behalf()** routines are the preferred method of testing access, this routine is useful for implementing operations like the conventional UNIX access function.

Permissions Required

The **sec_acl_get_access()** routine requires at least one permission of any kind on the object for which the access is to be returned.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_test_access(3sec)**, **sec_acl_test_access_on_behalf(3sec)**.

sec_acl_get_error_info

Purpose

Returns error information from an ACL handle

Synopsis

```
#include <dce/daclif.h>

error_status_t sec_acl_get_error_info(
        sec_acl_handle_t h);
```

Parameters

Input

h A handle referring to the target ACL. The handle is bound to the ACL with the **sec_acl_bind()** routine, which also specifies the name of the object to which the target ACL belongs.

Description

The **sec_acl_get_error_info()** routine returns error information from the specified ACL handle.

During a call to a routine in the *sec_acl* application programming interface (API), error codes received from the RPC runtime or other APIs are saved in the ACL handle and a corresponding error code from the *sec_acl* set is passed back by the ACL API. The **sec_acl_get_error_info()** routine returns the last error code stored in the ACL handle for those clients who need to know exactly what went wrong.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Return Values

This routine returns a value of type **error_status_t**, indicating the cause of the last error issued by the RPC runtime.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_invalid_handle
The ACL handle specified by **sec_acl_handle_t** is invalid.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_lookup(3sec)**, **sec_intro(3sec)**.

sec_acl_get_manager_types

Purpose

Lists the manager types of the ACLs protecting an object

Synopsis

```
#include <dce/daclif.h>

void sec_acl_get_manager_types(
        sec_acl_handle_t h,
        sec_acl_type_t sec_acl_type,
        unsigned32 size_avail,
        unsigned32 *size_used,
        unsigned32 *num_types,
        uuid_t manager_types[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

sec_acl_type

The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

size_avail An unsigned 32-bit integer containing the allocated length of the *manager_types[ ]* array.

Output

size_used An unsigned 32-bit integer containing the number of output entries returned in the *manager_types[ ]* array.

num_types An unsigned 32-bit integer containing the number of types returned in the *manager_types[ ]* array. This may be greater than *size_used* if there was not enough space allocated in the *manager_types[ ]* array for all the manager types.

manager_types[ ]
An array of length *size_avail* to contain UUIDs (of type **uuid_t**) identifying the different types of ACL managers protecting the target object.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_get_manager_types()** routine returns a list of the manager types of ACLs of type *sec_acl_type* that are protecting the object identified by *h*. For example, in addition to the regular file system ACL, a file representing the stable storage of some database could have an ACL manager that supported permissions allowing database updates only on certain days of the week.

ACL editors and browsers can use this operation to determine the ACL manager types that a particular reference monitor is using to protect a selected entity. Then, using the **sec_acl_get_printstring()** routine, they can determine how to format for display the permissions supported by a specific manager.

Permissions Required

The **sec_acl_get_manager_types()** routine requires at least one permission of any kind on the object for which the ACL manager types are to be returned.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_get_printstring(3sec)**, **sec_intro(3sec)**.

sec_acl_get_mgr_types_semantics

Purpose

Lists the manager types of the ACLs protecting an object

Synopsis

```
#include <dce/daclif.h>

void sec_acl_get_mgr_types_semantics(
        sec_acl_handle_t h,
        sec_acl_type_t sec_acl_type,
        unsigned32 size_avail,
        unsigned32 *size_used,
        unsigned32 *num_types,
        uuid_t manager_types[ ],
        sec_acl_posix_semantics_t posix_semantics[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

sec_acl_type

The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

size_avail An unsigned 32-bit integer containing the allocated length of the *manager_types[ ]* array.

Output

size_used An unsigned 32-bit integer containing the number of output entries returned in the *manager_types[ ]* array.

num_types An unsigned 32-bit integer containing the number of types returned in the *manager_types[ ]* array. This may be greater than *size_used* if there was not enough space allocated in the *manager_types[ ]* array for all the manager types.

manager_types[ ]
An array of length *size_avail* to contain UUIDs (of type **uuid_t**) identifying the different types of ACL managers protecting the target object.

posix_semantics[ ]
An array of POSIX semantics supported by each manager type with entries of type **sec_acl_posix_semantics_t**.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_get_mgr_types_semantics()** routine returns a list of the manager types of ACLs of type *sec_acl_type* that are protecting the object identified by *h*. For example, in addition to the regular file system ACL, a file representing the stable storage of some database could have an ACL manager that supported permissions allowing database updates only on certain days of the week.

ACL editors and browsers can use this operation to determine the ACL manager types that a particular reference monitor is using to protect a selected entity. Then, using the **sec_acl_get_printstring()** routine, they can determine how to format for display the permissions supported by a specific manager.

Permissions Required

The **sec_acl_get_mgr_types_semantics()** routine requires at least one permission of any kind on the object for which the ACL manager types are to be returned.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_get_printstring(3sec)**, **sec_intro(3sec)**.

sec_acl_get_printstring

Purpose

Returns printable ACL strings

Synopsis

```
#include <dce/daclif.h>

void sec_acl_get_printstring(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        unsigned32 size_avail,
        uuid_t *manager_type_chain,
        sec_acl_printstring_t *manager_info,
        boolean32 *tokenize,
        unsigned32 *total_num_printstrings,
        unsigned32 *size_used,
        sec_acl_printstring_t printstrings[ ],
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

size_avail An unsigned 32-bit integer containing the allocated length of the *printstrings[ ]* array.

Output

manager_type_chain
: If the target object ACL contains more than 32 permission bits, multiple manager types are used, one for each 32-bit wide ‘‘slice’’ of permissions. The UUID returned in *manager_type_chain* refers to the next ACL manager in the chain. If there are no more ACL managers for this ACL, **uuid_nil** is returned.

manager_info
: Provides a name and help string for the given ACL manager.

tokenize
: When FALSE, this variable indicates that the returned permission printstrings are unambiguous and therefore may be concatenated when printed without confusion. When TRUE, however, this property does not hold, and the strings need to be separated when printed or passed.

total_num_printstrings
: An unsigned 32-bit integer containing the total number of permission printstrings supported by this ACL manager type.

size_used
: An unsigned 32-bit integer containing the number of permission entries returned in the *printstrings[ ]* array.

printstrings[ ]
: An array of permission printstrings of type **sec_acl_printstring_t**. Each entry of the array is a structure containing the following three components:

  printstring A character string of maximum length **sec_acl_printstring_len** describing the printable representation of a specified permission.

  helpstring A character string of maximum length **sec_acl_printstring_help_len** containing some text that can be used to describe the specified permission.

  permissions A **sec_acl_permset_t** permission set describing the permissions that are represented with the companion printstring.

  The array consists of one such entry for each permission supported by the ACL manager identified by *manager_type*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_get_printstring()** routine returns an array of printable representations (called *printstrings*) for each permission bit or combination of permission bits the specified ACL manager supports. The ACL manager type specified must be one of the types protecting the object indicated by *h*.

In addition to returning the printstrings, this routine also returns instructions about how to print the strings. When the *tokenize* variable is set to FALSE, a printstring might be **r** or **w**, which could be concatenated in the display as **rw** without any confusion. However, when the *tokenize* variable is TRUE, it implies the printstrings might be of a form like **read** or **write**, which must be displayed separated by spaces or colons or something.

In any list of permission printstrings, there may appear to be some redundancy. ACL managers often define aliases for common permission combinations. By convention, however, simple entries should appear at the beginning of the *printstrings[ ]* array, and combinations should appear at the end.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not among those referenced by the input handle.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_get_manager_types(3sec)**, **sec_intro(3sec)**.

sec_acl_lookup

Purpose

Returns the ACL for an object

Synopsis

```
#include <dce/daclif.h>

void sec_acl_lookup(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        sec_acl_type_t sec_acl_type,
        sec_acl_list_t *sec_acl_list,
        error_status_t *status);
```

Parameters

Input

h
A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

sec_acl_type
The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

Output

sec_acl_list
: A pointer to the **sec_acl_list_t** structure to receive the complete access control list. An ACL contains a list of ACL entries, the UUID of the default cell where authentication takes place (foreign entries in the ACL contain the name of their home cell), and the UUID of the ACL manager to interpret the list.

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_lookup()** routine loads into memory a copy of an object's ACL corresponding to the specified manager type. The routine returns a pointer to the ACL. This routine is only used by ACL editors and browsers; an application would use **sec_acl_test_access()** or **sec_acl_test_access_on_behalf()** to process the contents of an ACL.

Permissions Required

The **sec_acl_lookup()** routine requires at least one permission of any kind on the object for which the ACL is to be returned.

Notes

The memory containing the **sec_acl_t** structure for each ACL is dynamically allocated. Use the **sec_acl_release()** routine to return each ACL's memory block to the pool when an application is finished with the ACLs.

Files

/usr/include/dce/daclif.idl
: The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

sec_acl_cant_allocate_memory
The requested operation requires more memory than is available.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_test_access(3sec)**, **sec_acl_test_access_on_behalf(3sec)**, **sec_intro(3sec)**.

sec_acl_release

Purpose

Releases ACL storage

Synopsis

```
#include <dce/daclif.h>

void sec_acl_release(
        sec_acl_handle_t h,
        sec_acl_t *sec_acl,
        error_status_t *status);
```

Parameters

Input

h — A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

sec_acl — A pointer to the complete ACL associated with the target object.

Output

status — A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_release()** routine releases any local storage associated with the ACL object, returning it to the pool. This is strictly a local operation (since the storage in question is local), and has no effect on the remote object or its ACL. The ACL handle is in the argument list only for consistency with other *sec_acl* routines.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_lookup(3sec)**, **sec_intro(3sec)**.

sec_acl_release_handle

Purpose

Removes an ACL handle

Synopsis

```
#include <dce/daclif.h>

void sec_acl_release_handle(
        sec_acl_handle_t *h,
        error_status_t *status);
```

Parameters

Input

h The handle to be removed. The handle is bound to the object to which the ACL belongs with the **sec_acl_bind()** routine.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_release_handle()** routine removes the specified handle. This is strictly a local operation, and has no effect on the remote object or its ACL.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_intro(3sec)**.

sec_acl_replace

Purpose

Replaces an ACL

Synopsis

```
#include <dce/daclif.h>

void sec_acl_replace(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        sec_acl_type_t sec_acl_type,
        sec_acl_list_t *sec_acl_list,
        error_status_t *status);
```

Parameters

Input

h
A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

sec_acl_type
The ACL type. The **sec_acl_type_t** data type distinguishes the various types of ACLs an object can possess for a given manager type. The possible values are as follows:

- **sec_acl_type_object**
- **sec_acl_type_default_object**
- **sec_acl_type_default_container**

sec_acl_list
The new ACL to use for the target object. This is represented by a pointer to the **sec_acl_list_t** structure containing the complete access control list. An ACL contains a list of ACL entries, the UUID of the default cell where authentication will take place (foreign entries in the ACL contain the name of their parent cell), and the UUID of the ACL manager to interpret the list.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_replace()** routine replaces the ACL indicated by the input handle with the information in the *sec_acl_list* parameter. ACLs are thought of as immutable, and in order to modify them, an editing application must read an entire ACL (using the **sec_acl_lookup()** routine), modify it as needed, and replace it using this routine.

Permissions Required

The **sec_acl_replace()** routine requires the **c** (control) permission on the object for which the ACL is to be replaced.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_lookup(3sec)**, **sec_intro(3sec)**.

sec_acl_test_access

Purpose

Tests access to an object

Synopsis

```
#include <dce/daclif.h>

boolean32 sec_acl_test_access(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        sec_acl_permset_t desired_permset,
        error_status_t *status);
```

Parameters

Input

h
A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

desired_permset
A permission set in **sec_acl_permset_t** form containing the desired privileges. This is a 32-bit set of permission flags supported by the manager type.

Output

status
A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_test_access()** routine determines if the specified ACL contains entries granting privileges to the calling process matching those in *desired_permset*. An application generally only inquires after the minimum set of privileges needed to accomplish a specific task.

Permissions Required

The **sec_acl_test_access()** routine requires at least one permission of any kind on the object for which the privileges are to be tested.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Return Values

The routine returns TRUE if the calling application program is authorized to access the target object with the privileges in *desired_permset*.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

Related Information

Functions: **sec_acl_bind(3sec)**, **sec_acl_test_access_on_behalf(3sec)**, **sec_intro(3sec)**.

sec_acl_test_access_on_behalf

Purpose

Tests access to an object on behalf of another process

Synopsis

```
#include <dce/daclif.h>

boolean32 sec_acl_test_access_on_behalf(
        sec_acl_handle_t h,
        uuid_t *manager_type,
        sec_id_pac_t *subject,
        sec_acl_permset_t desired_permset,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the target object. Use **sec_acl_bind()** to create this handle.

manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the object whose ACL is bound to the input handle. Use this parameter to distinguish them. Use **sec_acl_get_manager_types()** to acquire a list of the manager types protecting a given object.

subject A privilege attribute certificate (PAC) for the subject process. The PAC contains the name and UUID of the principal and cell of the subject process, as well as a list of any groups to which it belongs. The PAC also contains a flag (named **authenticated**). When set, it indicates that the certificate was obtained from an authenticated source. When not set, the certificate must not be trusted. (The field is FALSE when it was obtained from the **rpc_auth(3rpc)** layer and the protect level was set to **rpc_c_protect_level_none**. This indicates that no authentication

protocol was actually used in the remote procedure call; the identity was simply transmitted from the caller to the callee. If an authentication protocol was used, then the flag is set to TRUE.)

If a null PAC is passed, the subject is treated as an anonymous user, matching only the **any_other** and **unauthenticated** entries (if they exist) on the ACL.

A server uses **rpc_binding_inq_auth_client()** to acquire a certificate for the client process.

desired_permset
A permission set in **sec_acl_permset_t** form containing the desired privileges. This is a 32-bit set of permission flags supported by the manager type.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_acl_test_access_on_behalf()** routine determines if the specified ACL contains entries that grant the privileges specified in *desired_permset* to the subject process. An application generally inquires about only the minimum set of privileges needed to accomplish a specific task.

Permissions Required

The **sec_acl_test_access_on_behalf()** routine requires at least one permission of any kind on the object for which the privileges are to be tested. Both the calling process and the identified subject must have permission on the object.

Note: This operation is obsolete, but is documented for backward compatibility. **sec_id_pac_t** is no longer the data structure used for identities (for further information, see the **sec_cred_*(3sec)** routines), and delegation subsumes the functionality that the **sec_acl_test_access_on_behalf()** routine was originally intended to provide. ACL managers do not have to implement the server side of this functionality to be DCE compliant, and therefore clients should not rely on its being available in servers.

Files

/usr/include/dce/daclif.idl
The **idl** file from which **dce/daclif.h** was derived.

Return Values

If the routine completes successfully (with a completion status of **error_status_ok**) it returns a value of

- TRUE, if the caller has any access (at least one permission of any kind), and the subject has the *desired_permset* privileges.
- FALSE, if both the caller and the subject have any access, but the subject does not have the *desired_permset* privileges.

If the routine does not complete successfully, it returns a bad completion status code and a return value of FALSE.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_acl_unknown_manager_type
The manager type selected is not an available option.

error_status_ok
The call was successful.

sec_acl_not_implemented
Requested operation is not implemented in this version of DCE.

Related Information

Functions: **rpc_binding_inq_auth_client(3rpc)**, **sec_acl_bind(3sec)**, **sec_acl_test_access(3sec)**, **sec_intro(3sec)**.

sec_attr_trig_query

Purpose

Reads attributes coded with an attribute trigger type of query

Synopsis

```
#include <dce/sec_attr_trig.h>

void sec_attr_trig_query (
        handle_t h,
        sec_attr_component_name_t cell_name,
        sec_attr_component_name_t component_name,
        sec_attr_trig_cursor_t *cursor,
        unsigned32 num_attr_keys,
        unsigned32 space_avail,
        sec_attr_t attr_keys[ ],
        unsigned32 *num_returned,
        sec_attr_t attrs[ ],
        sec_attr_trig_timeval_sec_t time_to_live[ ],
        unsigned32 *num_left,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the trigger server to be accessed Use the trigger binding information specified in the attribute encoding to acquire a bound handle.

cell_name A value of **sec_attr_component_name_t** that identifies the cell in which the object whose attribute is to be accessed resides. Supply a NULL *cell_name* to specify the local cell (**/.:**).

component_name
A value of **sec_attr_component_name_t** that identifies the name of the object whose attribute is to be accessed. If *cell_name* specifies a foreign

cell, *component_name* is interpreted as a UUID in string format since the caller of this interface knows only the UUID, not the name, of the foreign principal.

num_attr_keys
An unsigned 32-bit integer that specifies the number of elements in the *attr_keys[ ]* array. This integer must be greater than 0 (zero).

space_avail An unsigned 32-bit integer that specifies the size of the *attr_keys[ ]* array.

attr_keys[ ] An array of values of type **sec_attr_t**. For each attribute instance, the **sec_attr_t** array contains an *attr_id* (a UUID of type **uuid_t**) to identify the attribute to be queried and an *attr_value*. *attr_value* can be used to pass in optional information required by the attribute trigger query. If no additional information is to be passed, set *attr_value* to **sec_attr_enc_void**. This is actually accomplished by setting the **sec_attr_encoding_t** data type to **sec_attr_enc_void**.

The size of the *attr_keys[ ]* array is determined by *num_attr_keys*.

Input/Output

cursor A pointer to a cursor of type **sec_attr_trig_cursor_t**. As an input parameter, *cursor* can be initialized (by the server) or uninitialized. If the cursor is uninitialized, the cursor begins processing the query at the first attribute that satisfies the search criteria. As an output parameter, *cursor* is positioned past the attributes returned in this call.

Output

num_returned
A pointer to an unsigned 32-bit integer that specifies the number of attribute instances returned in the *attr_keys[ ]* array.

attrs[ ] An array of values of type **sec_attr_t**. The size of this array is determined by the *space_avail* parameter and the length by the *num_returned* parameter.

time_to_live[ ]
An array of values of type **sec_attr_trig_timeval_sec_t**. For each attribute in the *attrs*[ ] array, The *time_to_live[ ]* array specifies the time in seconds that the attribute can be safely cached.

num_left A pointer to an unsigned 32-bit integer that supplies the number of attributes found but not returned because of space constraints in the *attrs*[ ] buffer.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_attr_trig_query()** routine reads attributes coded with a attribute trigger type of query.

The **sec_attr_trig_query()** routine is called by the DCE attribute lookup code for all schema entries that specify a query attribute trigger (**sec_attr_trig_type_query** specified with the **sec_attr_trig_type_flags_t** data type). The attribute query code passes the **sec_attr_trig_query()** input parameters to a user-written query attribute trigger server and receives the output parameters back from the server. Although generally this routine is not called directly, this reference page is provided for users who are writing the attribute trigger servers that will receive **sec_attr_trig_query()** input and supply its output.

Multivalued attributes are returned as independent attribute instances sharing the same attribute UUID. A read of an attribute set returns all instances of members of the set; the attribute set instance is not returned.

For objects in the local cell, set the *cell_name* parameter to **null**, and the *component_name* parameter to specify the object's name.

For objects in a foreign cell, set the *cell_name* parameter to identify the name of the foreign cell, and the *component_name* parameter to the UUID in string format that identifies the object in the foreign cell.

The *num_left* parameter contains the number of attributes that were found but could not be returned because of space constraints of the *attrs[ ]* array. (Note that this number may be inaccurate if the target server allows updates between successive queries.) To obtain all of the remaining attributes, set the size of the *attrs[ ]* array so that it is large enough to hold the number of attributes listed in *num_left*.

Files

/usr/include/dce/sec_attr_trig.idl
The **idl** file from which **dce/sec_attr_trig.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

not_all_available

unauthorized

error_status_ok

Related Information

Functions: **sec_attr_trig_cursor_init**, **sec_attr_trig_update(3sec)**, **sec_intro(3sec)**.

sec_attr_trig_update

Purpose

For attributes coded with an attribute trigger type of update, passes attribute updates to an update attribute trigger server for evaluation

Synopsis

```
#include <dce/sec_attr_trig.h>

void sec_attr_trig_update (
        handle_t h,
        sec_attr_component_name_t cell_name,
        sec_attr_component_name_t component_name,
        unsigned32 num_to_write,
        unsigned32 space_avail,
        sec_attr_t in_attrs[ ],
        unsigned32 *num_returned,
        sec_attr_t out_attrs[ ],
        unsigned32 *num_left,
        signed32 *failure_index,
        error_status_t *status);
```

Parameters

Input

h A handle referring to the trigger server to be accessed. Use the trigger binding information specified in the attribute encoding to acquire a bound handle.

cell_name A value of **sec_attr_component_name_t** that identifies the cell in which the object whose attribute is to be accessed resides. Supply a NULL *cell_name* to specify the local cell (**/.:**).

component_name
A value of **sec_attr_component_name_t** that identifies the name of the object whose attribute is to be accessed. If *cell_name* specifies a foreign

cell, *component_name* is interpreted as a UUID in string format since the caller of this interface knows only the UUID, not the name, of the foreign principal.

num_to_write
An unsigned 32-bit integer that specifies the number of elements in the *in_attrs* array. This integer must be greater than 0 (zero).

space_avail An unsigned 32-bit integer that specifies the size of the *out_attrs* array.

in_attrs[ ] An array of values of type **sec_attr_t** that specifies the attribute instances to be written. The size of *in_attrs[ ]* is determined by *num_to_write*.

Output

num_returned
A pointer to an unsigned 32-bit integer that specifies the number of attribute instances returned in the *out_attrs[ ]* array.

out_attrs[ ] An array of values of type **sec_attr_t**. These values, supplied by the update attribute trigger server, are in a form suitable for storage in the registry database.

num_left A pointer to an unsigned 32-bit integer that supplies the number of attributes that were found but not returned because of space constraints in the *out_attrs[ ]* buffer.

failure_index
In the event of an error, *failure_index* is a pointer to the element in the *in_attrs[ ]* array that caused the update to fail. If the failure cannot be attributed to a specific attribute, the value of *failure_index* is −1.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_attr_trig_update()** routine passes attributes coded with an attribute trigger type of update to a user-written update attribute trigger server for evaluation before the updates are made to the registry.

Although generally this routine it is not called directly, this reference page is provided for users who are writing the attribute trigger servers that will receive **sec_attr_trig_update()** input and supply its output.

The **sec_attr_trig_update()** routine is called by the DCE attribute update code for all schema entries that specify an update attribute trigger (**sec_attr_trig_type_update** specified with the **sec_attr_trig_type_flags_t** data type). The attribute update code passes the **sec_attr_trig_update()** input parameters to a user-written update attribute trigger server and receives the output parameters back from the server. The attribute trigger server is responsible for evaluating the semantics of the entry in order to reject or accept it, and the attribute trigger server may even make changes in the output it sends back to the update code to ensure the entry adheres to the semantics. The output received from the attribute trigger server is in a form to be stored in the registry. (Note that update attribute trigger servers do not store attribute values. Attribute values are stored in the registry database.)

This is an atomic operation: if the update of any attribute in the array fails to pass the evaluation, all updates are aborted. The attribute causing the update to fail is identified in *failure_index*. If the failure cannot be attributed to a given attribute, *failure_index* contains −1.

For objects in the local cell, set the *cell_name* parameter to **null**, and the *component_name* parameter to specify the object's name.

For objects in a foreign cell, set the *cell_name* parameter the the name of the foreign cells, and the *component_name* parameter to specify the UUID in string format that identifies the object in the foreign cell.

Files

/usr/include/dce/sec_attr_trig.idl
The **idl** file from which **dce/sec_attr_trig.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

database read only

server unavailable

invalid/unsupported attribute type

invalid encoding type

value not unique

site read only

unauthorized

error_status_ok

Related Information

Functions: **sec_attr_trig_query(3sec)**, **sec_intro(3sec)**.

sec_attr_util_alloc_copy

Purpose

Allocates the necessary subfields of the destination **sec_attr_t** and copies the corresponding data from the source **sec_attr_t**

Synopsis

```
#include <dce/sec_attr_util.h>

void sec_attr_util_alloc_copy (
        void *(*allocate) (unsigned32 size),
        sec_attr_t *from,
        sec_attr_t *to,
        error_status_t *status);
```

Parameters

Input

*(*allocate)* (**unsigned32** *size*)
: A caller-specified allocate routine (such as **rpc_ss_allocate()**) used to allocate resources for the output *to* parameter. Set to NULL to use the default **malloc()** routine.

**from*
: A pointer to a **sec_attr_t** that is the source to be copied from.

Output

**to*
: A pointer to the target **sec_attr_t** that contains subfields allocated, if necessary, by the caller-specified allocate routine and data copied from the source **sec_attr_t** specified by *from*.

**status*
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

sec_attr_util_alloc_copy(3sec)

Description

The **sec_attr_util_alloc_copy()** routine allocates memory for the subfields of the target **sec_attr_t**, if necessary, and copies data from the source **sec_attr_t** to the target **sec_attr_t**.

Use the **sec_attr_util_free()** routine to free the memory allocated by this routine. If a nonnull allocate routine was input to **sec_attr_util_alloc_copy()**, then a corresponding free routine must be input to the **sec_attr_util_free()** routine.

Files

/usr/include/dce/sec_attr_util.idl
The **idl** file from which **dce/sec_attr_util.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_implemented

error_status_ok

Related Information

Functions: **sec_attr_util_free(3sec)**, **sec_attr_util_inst_free_ptrs(3sec)**, **sec_attr_util_inst_free(3sec)**.

sec_attr_util_free

Purpose

Frees nonnull pointers in a **sec_attr_t** with an input deallocate routine

Synopsis

#include <dce/sec_attr_util.h>

void sec_attr_util_free(
void *(*deallocate)* **(void** **ptr***),**
sec_attr_t * *attr***);**

Parameters

Input/Output

(**deallocate*)**(void** **ptr***)**
A caller-specified memory deallocate routine. If set to NULL, the default **free()** is used.

**attr*
As input, a pointer to a **sec_attr_t** for which memory should be deallocated. As output, a pointer to the **sec_attr_t** with subfields, if any, deallocated and set to NULL.

Description

The **sec_attr_util_free()** routine uses the input *deallocate* routine to free memory allocated to a **sec_attr_t** by **sec_attr_util_alloc_copy()**. With an input value of NULL for *deallocate*, the **sec_attr_util_free** routine behaves identically to **sec_attr_util_inst_free_ptrs**.

Files

/usr/include/dce/sec_attr_util.idl
The **idl** file from which **dce/sec_attr_util.h** was derived.

Related Information

Functions: **sec_attr_util_alloc_copy(3sec)**, **sec_attr_util_inst_free_ptrs(3sec)**, **sec_attr_util_inst_free(3sec)**.

sec_attr_util_inst_free

Purpose

Frees nonnull pointers in a **sec_attr_t** and the pointer to the **sec_attr_t** itself

Synopsis

```
#include <dce/sec_attr_util.h>

void sec_attr_util_inst_free (
        sec_attr_t **sec_attr_p);
```

Parameters

Input/Output

***sec_attr_p* As input, the address of an allocated pointer to a potentially initialized **sec_attr_t**. As output, the address of a deallocated pointer that has been set to NULL.

Description

The **sec_attr_util_inst_free()** routine frees each nonnull pointer in a **sec_attr_t** pointed to by **sec_attr_p*. The **sec_attr_p* itself is also freed and set to NULL. A partially initialized **sec_attr_t** is handled correctly .

The **sec_attr_util_inst_free()** routine is useful for freeing the resources of dynamically allocated **sec_attr_t**s and their subfields.

Note that most DCE client application programming interfaces (APIs) that return **sec_attr_t**s allocate only subfields, and not the **sec_attr_t** itself. Use **sec_attr_util_inst_free_ptrs** instead of **sec_attr_util_inst_free** to free attribute resources allocated by such APIs.

Files

/usr/include/dce/sec_attr_util.idl
The **idl** file from which **dce/sec_attr_util.h** was derived.

Related Information

Functions: **sec_attr_util_inst_free_ptrs(3sec)**.

sec_attr_util_inst_free_ptrs

Purpose

Frees nonnull pointers in a **sec_attr_t**

Synopsis

```
#include <dce/sec_attr_util.h>

void sec_attr_util_inst_free_ptrs (
        sec_attr_t *sec_attr_p);
```

Parameters

Input/Output

**sec_attr_p*

As input, a pointer to an allocated and potentially initialized **sec_attr_t**. As output, a pointer to a **sec_attr_t** with internal pointers freed and set to NULL. The **sec_attr_t** itself is not freed.

Description

The **sec_attr_util_inst_free_ptrs()** routine frees and sets to NULL each nonnull pointer in a **sec_attr_t** pointed to by *sec_attr_p*. The **sec_attr_t** itself is not freed. The **sec_attr_t** may have been only partially initialized.

Files

/usr/include/dce/sec_attr_util.idl

The **idl** file from which **dce/sec_attr_util.h** was derived.

Related Information

Functions: **sec_attr_util_inst_free(3sec)**.

sec_attr_util_sch_ent_free

Purpose

Frees nonnull pointers in a **sec_attr_schema_entry_t** and the pointer to the **sec_attr_schema_entry_t** itself

Synopsis

```
#include <dce/sec_attr_util.h>

void sec_attr_util_sch_ent_free (
        sec_attr_schema_entry_t **sec_sch_entry_p);
```

Parameters

Input/Output

***sec_sch_entry_p*

As input, the address of an allocated pointer to a potentially initialized **sec_attr_schema_entry_t**. As output, the address of a deallocated pointer that has been set to NULL.

Description

The **sec_attr_util_sch_ent_free()** routine frees each nonnull pointer in a **sec_attr_schema_entry_t** pointed to by a **sec_sch_entry_p*. The **sec_sch_entry_p* itself is also freed and set to NULL. A partially initialized **sec_attr_schema_entry_t** is handled correctly

Files

/usr/include/dce/sec_attr_util.idl

The **idl** file from which **dce/sec_attr_util.h** was derived.

Related Information

Functions: **sec_attr_util_sch_ent_free_ptrs(3sec)**.

sec_attr_util_sch_ent_free_ptrs

Purpose

Frees nonnull pointers in a **sec_attr_schema_entry_t**

Synopsis

```
#include <dce/sec_attr_util.h>

void sec_attr_util_sch_ent_free_ptrs (
        sec_attr_schema_entry_t *sec_sch_entry_p);
```

Parameters

Input/Output

**sec_sch_entry_p*

As input, a pointer to an allocated and potentially initialized **sec_attr_schema_entry_t**. As output, a pointer to a **sec_attr_schema_entry_t** with internal pointers freed and set to NULL.

Description

The **sec_attr_util_sch_ent_free_ptrs()** routine frees and sets to NULL each nonnull pointer in a **sec_attr_schema_entry_t** pointed to by *sec_sch_entry_p*. The *sec_sch_entry_p* itself is not freed. A partially initialized **sec_attr_schem_entry_t** is handled correctly.

Files

/usr/include/dce/sec_attr_util.idl

The **idl** file from which **dce/sec_attr_util.h** was derived.

Related Information

Functions: **sec_attr_util_sch_ent_free(3sec)**.

sec_cred_free_attr_cursor

Purpose

Frees the local resources allocated to a **sec_attr_cursor_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_free_attr_cursor (
        sec_cred_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor As input, a pointer to a **sec_cred_attr_cursor_t** whose resources are to be freed. As output a pointer to an initialized **sec_cred_attr_cursor_t** with allocated resources freed.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_free_attr_cursor()** routine frees the resources assoicated with a cursor of type **sec_cred_attr_cursor_t** used by the **sec_cred_get_extended_attrs()** call.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

Related Information

Functions: **sec_cred_get_extended_attrs(3sec)**, **sec_cred_initialize_attr_cursor(3sec)**, **sec_intro(3sec)**.

sec_cred_free_cursor

Purpose

Releases local resources allocated to a **sec_cred_cursor_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_free_cursor (
        sec_cred_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor As input, a **sec_cred_cursor_t** whose resources are to be freed. As output, a **sec_cred_cursor_t** whose resources are freed.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_free_cursor()** routine releases local resources allocated to a **sec_cred_cursor_t** used by the **sec_cred_get_delegate()** call.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_memory

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_initialize_cursor(3sec)**, **sec_intro(3sec)**.

sec_cred_free_pa_handle

Purpose

Frees the local resources allocated to a privilege attribute handle of type **sec_cred_pa_handle_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_free_pa_handle (
        sec_cred_pa_handle__t *pa_handle,
        error_status_t *status);
```

Parameters

Input/Output

pa_handle As input, a pointer to a **sec_cred_pa_handle_t** whose resources are to be freed. As output a pointer to a **sec_cred_pa_handle_t** with allocated resources freed.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_free_pa_handle()** routine frees the resources assoicated with a privilege attribute handle of type **sec_cred_pa_handle_t** used by the **sec_cred_get_initiator()** and **sec_cred_get_delegate()** calls.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_authz_session_info

Purpose

Returns session-specific information that represents an authenticated client's credentials

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_get_authz_session_info(
        rpc_authz_cred_handle_t callers_identity,
        uuid_t *session_id,
        sec_timeval_t *session_expiration,
        error_status_t *status);
```

Parameters

Input

callers_identity
: A credential handle of type **rpc_authz_cred_handle_t**. This handle is supplied as output of the **rpc_binding_inq_auth_caller()** call.

Output

session_ID
: A pointer to a **uuid_t** that identifies the client's DCE authorization session.

session_expiration
: A pointer to a **sec_timeval_t** that specifies the expiration time of the authenticated client's credentials.

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_get_authz_session_info()** routine retrieves session-specific information that represents the credentials of authenticated client specified by *callers_identity*. If the client is a member of a delegation chain, the information represents the credentials of all members of the chain.

The information can aid application servers in the construction of identity-based caches. For example, it could be used as a key into a cache of previously allocated delegation contexts and thus avoid the overhead of allocating a new login context on every remote operation. It could also be used as a key into a table of previously computed authorization decisions.

Before you execute this call, you must execute an **rpc_binding_inq_auth_caller()** call to obtain an **rpc_authz_cred_handle_t** for the *callers_identity* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_authz_cannot_comply

error_status_ok

Related Information

Functions: **rpc_binding_inq_auth_caller(rpc)**, **sec_intro(3sec)**.

sec_cred_get_client_princ_name

Purpose

Returns the principal name associated with a credential handle

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_get_client_princ_name(
        rpc_authz_cred_handle_t callers_identity,
        unsigned_char_p_t *client_princ_name,
        error_status_t *status);
```

Parameters

Input

callers_identity
: A handle of type **rpc_authz_cred_handle_t** to the credentials for which to return the principal name. This handle is supplied as output of the **rpc_binding_inq_auth_caller()** call.

Output

client_princ_name
: A pointer to the principal name of the server's RPC client.

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_get_client_princ_name()** routine extracts the principal name associated with the credentials identified by *callers_pas*.

Before you execute **sec_cred_get_client_princ_name()**, you must execute an **rpc_binding_inq_auth_caller()** call to obtain an **rpc_authz_cred_handle_t** for the *callers_identity* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_authz_cannot_comply

error_status_ok

Related Information

Functions: **rpc_binding_inq_auth_caller(3sec)**, **sec_intro(3sec)**.

sec_cred_get_deleg_restrictions

Purpose

Returns delegate restrictions from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_restriction_set_t *sec_cred_get_deleg_restrictions(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A value of type **sec_cred_pa_handle_t** that provides a handle to a principal's privilege attributes. This handle is supplied as output of the **sec_cred_get_initiator()** call, the **sec_cred_get_delegate()** call and the **sec_login_cred** calls.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_deleg_restrictions()** routine extracts delegate restrictions from the privilege attribute handle identified by *callers_pas*. The restrictions are returned in a **sec_id_restriction_set_t**.

Before you execute **sec_cred_get_pa_data()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_delegate

Purpose

Returns a handle to the privilege attributes of an intermediary in a delegation chain

Synopsis

```
#include <dce/sec_cred.h>

sec_cred_pa_handle_t sec_cred_get_delegate(
        rpc_authz_cred_handle_t callers_identity,
        sec_cred_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input

callers_identity
: A handle of type **rpc_authz_cred_handle_t**. This handle is supplied as output of the **rpc_binding_inq_auth_caller()** call.

Input/Output

cursor
: As input, a pointer to a cursor of type **sec_cred_cursor_t** that has been initialized by the **sec_cred_initialize_cursor()** call. As an output parameter, *cursor* is a pointer to a cursor of type **sec_attr_srch_cursor_t** that is positioned past the principal whose privilege attributes have been returned in this call.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_delegate()** routine returns a handle to the the privilege attributes of an intermediary in a delegation chain that performed an authenticated RPC operation.

This call is used by servers. Clients use the **sec_login_cred_get_delegate()** routine to return the privilege attribute handle of an intermediary in a delegation chain.

The credential handle identified by *callers_identity* contains authentication and authorization information for all delegates in the chain. This call returns a handle (**sec_cred_pa_handle_t**) to the privilege attributes of one of the delegates in the binding handle. The **sec_cred_pa_handle_t** returned by this call is used in other **sec_cred_get_*** calls to obtain privilege attribute information for a single delegate.

To obtain the privilege attributes of each delegate in the credential handle identified by *callers_identity*, execute this call until the message **sec_cred_s_no_more_entries** is returned.

Before you execute **sec_cred_get_delegate()**, you must execute

- An **rpc_binding_inq_auth_caller()** call to obtain an **rpc_authz_cred_handle_t** for the *callers_identity* parameter.
- A **sec_cred_initialize_cursor()** call to initialize a cursor of type **sec_cred_cursor_t**.

Use the **sec_cred_free_pa_handle()** all to free the resources associated with the **sec_cred_pa_handle_t**.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_auth_handle

sec_cred_s_invalid_cursor

sec_cred_s_no_more_entries

error_status_ok

Related Information

Functions: **rpc_binding_inq_auth_caller(3rpc)**, **sec_cred_free_pa_handle()**, **sec_cred_get_deleg_restrictions(3sec)**, **sec_cred_get_delegation_type(3sec),** **sec_cred_get_extended_attrs(3sec)**, **sec_cred_get_opt_restrictions(3sec)**, **sec_cred_get_pa_date**, **sec_cred_get_req_restrictions(3sec)**, **sec_cred_get_tgt_restrictions(3sec)**, **sec_cred_get_v1_pac(3sec)** **sec_cred_initialize_cursor(3sec)**, **sec_intro(3sec)**.

sec_cred_get_delegation_type

Purpose

Returns the delegation type from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_delegation_type_t *sec_cred_get_delegation_type(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A value of type **sec_cred_pa_handle_t** that provides a handle to a principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_delegation_type ()** routine extracts the delegation type from the privilege attribute handle identified by *callers_pas* and returns it in a **sec_id_delegation_type_t**.

Before you execute **sec_cred_get_delegation_type()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_extended_attrs

Purpose

Returns extended attributes from a privilege handle

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_get_extended_attrs(
        sec_cred_pa_handle_t callers_pas,
        sec_cred_attr_cursor_t *cursor,
        sec_attr_t *attr,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to the caller's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Input/Output

cursor A cursor of type **sec_cred_attr_cursor_t** that has been initialized by the **sec_cred_initialize_attr_cursor()** routine. As input *cursor* must be initialized. As output, *cursor* is positioned at the first attribute after the returned attribute.

Output

attr A pointer to a value of **sec_attr_t** that contains extended registry attributes.

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

sec_cred_get_extended_attrs(3sec)

Description

The **sec_cred_get_extended_attrs()** routine extracts extended registry initialized from the privilege attribute handle identified by *callers_pas*.

Before you execute call, you must execute

- A **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.
- A **sec_cred_initialize_attr_cursor()** to initialize a **sec_attr_t**.

To obtain all the extended registry attributes in the privilege attribute handle, repeat **sec_cred_get_extended_attrs()** calls until the status message **no_more_entries_available** is returned.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

sec_cred_s_invalid_cursor

sec_cred_s_no_more_entries

error_status_ok

Related Information

Functions: **sec_cred_get_initiator(3sec)**, **sec_cred_get_delegate(3sec)**, **sec_cred_initialize_attr_cursor(3sec)**, **sec_intro(3sec)**.

sec_cred_get_initiator

Purpose

Returns the privilege attributes of the initiator of a delegation chain

Synopsis

```
#include <dce/sec_cred.h>

sec_cred_pa_handle_t sec_cred_get_initiator(
        rpc_authz_cred_handle_t callers_identity,
        error_status_t *status);
```

Parameters

Input

callers_identity
: A credential handle of type **rpc_authz_cred_handle_t**. This handle is supplied as output of the **rpc_binding_inq_auth_caller()** call.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_initiator()** routine returns a handle to the the privilege attributes of the initiator of a delegation chain that performed an authenticated RPC operation.

The credential handle identified by *callers_identity* contains authentication and authorization information for all delegates in the chain. This call returns a handle (**sec_cred_pa_handle_t**) to the privilege attributes of the client that initiated the delegation chain. The **sec_cred_pa_handle_t** returned by this call is used in other **sec_cred_get...** calls to obtain privilege attribute information for the initiator.

Before you execute **sec_cred_get_initiator()**, you must execute an **rpc_binding_inq_auth_caller()** call to obtain an **rpc_authz_cred_handle_t** for the *callers_identity* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_auth_handle

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **rpc_binding_inq_auth_caller(3rpc)**, **sec_cred_get_deleg_restrictions(3sec)**, **sec_cred_get_delegation_type(3sec)**, **sec_cred_get_extended_attrs(3sec)**, **sec_cred_get_opt_restrictions(3sec)**, **sec_cred_get_pa_date**, **sec_cred_get_req_restrictions(3sec)**, **sec_cred_get_tgt_restrictions(3sec)**, **sec_cred_get_v1_pac(3sec)**.

sec_cred_get_opt_restrictions

Purpose

Returns optional restrictions from a privilege handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_opt_req_t *sec_cred_get_opt_restrictions(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to a principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_opt_restrictions ()** routine extracts optional restrictions from the privilege attribute handle identified by *callers_pas* and returns them in a **sec_id_restriction_set_t**.

Before you execute **sec_cred_get_pa_data()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_pa_data

Purpose

Returns identity information from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_pa_t *sec_cred_get_pa_data(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to a principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_pa_data()** routine extracts identity information from the privilege attribute handle specified by *callers_pas* and returns it in a **sec_id_pa_t**. The identity information includes an identifier of the princpal's locall cell and the principal's local and foreign group sets.

Before you execute **sec_cred_get_pa_data()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_req_restrictions

Purpose

Returns required restrictions from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_opt_req_t *sec_cred_get_req_restrictions(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to a principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_req_restrictions()** routine extracts required restrictions from the privilege attribute handle identified by *callers_pas* and returns them in a **sec_id_opt_req_t**.

Before you execute **sec_cred_get_req_restrictions()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_tgt_restrictions

Purpose

Returns target restrictions from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_restriction_set_t *sec_cred_get_tgt_restrictions(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to a principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_tgt_restrictions()** routine extracts target restrictions from the privilege attribute handle identified by *callers_pas* and returns them in a **sec_id_restriction_set_t**.

Before you execute **sec_cred_get_tgt_restrictions()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_get_v1_pac

Purpose

Returns pre-1.1 PAC from a privilege attribute handle

Synopsis

```
#include <dce/sec_cred.h>

sec_id_pac_t *sec_cred_get_v1_pac(
        sec_cred_pa_handle_t callers_pas,
        error_status_t *status);
```

Parameters

Input

callers_pas A handle of type **sec_cred_pa_handle_t** to the principal's privilege attributes. This handle is supplied as output of either the **sec_cred_get_initiator()** call or **sec_cred_get_delegate()** call.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**.

Description

The **sec_cred_get_v1_pac()** routine extracts the privilege attributes from a pre-1.1 PAC for the privilege attribute handle specified by *callers_pas* and returns them in a **sec_id_pa_t**.

Before you execute **sec_cred_get_v1_pac()**, you must execute a **sec_cred_get_initiator()** or **sec_cred_get_delegate()** call to obtain a **sec_cred_pa_handle_t** for the *callers_pas* parameter.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_pa_handle

error_status_ok

Related Information

Functions: **sec_cred_get_delegate(3sec)**, **sec_cred_get_initiator(3sec)**, **sec_intro(3sec)**.

sec_cred_initialize_attr_cursor

Purpose

Initializes a **sec_attr_cursor_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_initialize_attr_cursor (
        sec_cred_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor As input, a pointer to a **sec_cred_attr_cursor_t** to be initialized. As output a pointer to an initialized **sec_cred_attr_cursor_t**.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_initialize_attr_cursor()** routine allocates and initializes a cursor of type **sec_cred_attr_cursor_t** for use with the **sec_cred_get_extended_attrs()** call. Use the **sec_cred_free_attr_cursor()** call to free the resources allocated to *cursor*.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_memory

error_status_ok

Related Information

Functions: **sec_cred_free_attr_cursor()**, **sec_cred_get_extended_attrs(3sec)**, **sec_intro(3sec)**.

sec_cred_initialize_cursor

Purpose

Initializes a **sec_cred_cursor_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_cred_initialize_cursor (
        sec_cred_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor As input, a **sec_cred_cursor_t** to be initialized. As output, an initialized **sec_cred_cursor_t**.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_initialize_cursor()** routine initializes a cursor of type **sec_cursor_t** for use with the **sec_cred_get_delegate()** call. Use the **sec_cred_free_cursor()** call to free the resources allocated to *cursor*.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_memory

error_status_ok

Related Information

Functions: **sec_cred_free_cursor(3sec)**, **sec_cred_get_delegate(3sec)**, **sec_intro(3sec)**.

sec_cred_is_authenticated

Purpose

Returns TRUE if the supplied credentials are authenticated, and FALSE if they are not

Synopsis

```
#include <dce/sec_cred.h>

boolean32 sec_cred_is_authenticated(
        rpc_authz_cred_handle_t callers_identity,
        error_status_t *status);
```

Parameters

Input

callers_identity
: A handle of type **rpc_authz_cred_handle_t** to the credentials to check for authentication. This handle is supplied as output of the **rpc_binding_inq_auth_caller()** call.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_cred_is_authenticated()** routine returns TRUE if the credentials identified by *callers_identity* are authenticated or FALSE if they are not.

Before you execute this call, you must execute an **rpc_binding_inq_auth_caller()** call to obtain an **rpc_authz_cred_handle_t** for the *callers_identity* parameter.

Files

/usr/include/dce/sec_cred.idl	The **idl** file from which **dce/sec_cred.h** was derived.

Return Values

The routine returns **true** if the credentials are authenticated; **false** if they are not.

Related Information

Functions: **rpc_binding_inq_auth_caller(3rpc)**, **sec_intro(3sec)**.

sec_id_gen_group

Purpose

Generates a global name from cell and group UUIDs

Synopsis

```
#include <dce/secidmap.h>

void sec_id_gen_group(
        sec_rgy_handle_t context,
        uuid_t *cell_idp,
        uuid_t *group_idp,
        sec_rgy_name_t global_name,
        sec_rgy_name_t cell_namep,
        sec_rgy_name_t group_namep,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

cell_idp A pointer to the UUID of the home cell of the group whose name is in question.

group_idp A pointer to the UUID of the group whose name is in question.

Output

global_name The global (full) name of the group in **sec_rgy_name_t** form.

cell_namep The name of the group's home cell in **sec_rgy_name_t** form.

group_namep
The local (with respect to the home cell) name of the group in **sec_rgy_name_t** form.

status A pointer to the completion status. On successful completion, the function returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_id_gen_group()** routine generates a global name from input cell and group UUIDs. For example, given a UUID specifying the cell **/.../world/hp/brazil**, and a UUID specifying a group resident in that cell named **writers**, the routine would return the global name of that group, in this case, **/.../world/hp/brazil/writers**. It also returns the simple names of the cell and group, translated from the UUIDs.

The routine will not produce translations to any name for which a NULL pointer has been supplied.

Files

/usr/include/dce/secidmap.idl
The **idl** file from which **dce/secidmap.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_id_e_name_too_long
The name is too long for current implementation.

sec_id_e_bad_cell_uuid
The cell UUID is not valid.

sec_rgy_object_not_found
The registry server could not find the specified group.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_id_gen_name(3sec)**, **sec_id_parse_group(3sec)**, **sec_id_parse_name(3sec)**, **sec_intro(3sec)**.

sec_id_gen_name

Purpose

Generates a global name from cell and principal UUIDs

Synopsis

```
#include <dce/secidmap.h>

void sec_id_gen_name(
        sec_rgy_handle_t context,
        uuid_t *cell_idp,
        uuid_t *princ_idp,
        sec_rgy_name_t global_name,
        sec_rgy_name_t cell_namep,
        sec_rgy_name_t princ_namep,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

cell_idp A pointer to the UUID of the home cell of the principal whose name is in question.

princ_idp A pointer to the UUID of the principal whose name is in question.

Output

global_name The global (full) name of the principal in **sec_rgy_name_t** form.

cell_namep
The name of the principal's home cell in **sec_rgy_name_t** form.

princ_namep
The local (with respect to the home cell) name of the principal in **sec_rgy_name_t** form.

status A pointer to the completion status. On successful completion, the function returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_id_gen_name()** routine generates a global name from input cell and principal UUIDs. For example, given a UUID specifying the cell **/.../world/hp/brazil**, and a UUID specifying a principal resident in that cell named **writers/tom**, the routine would return the global name of that principal, in this case, **/.../world/hp/brazil/writers/tom**. It also returns the simple names of the cell and principal, translated from the UUIDs.

The routine will not produce translations to any name for which a NULL pointer has been supplied.

Permissions Required

The **sec_id_gen_name()** routine requires at least one permission of any kind on the account associated with the input cell and principal UUIDs.

Files

/usr/include/dce/secidmap.idl
The **idl** file from which **dce/secidmap.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_id_e_name_too_long
The name is too long for current implementation.

sec_id_e_bad_cell_uuid
The cell UUID is not valid.

sec_rgy_object_not_found
The registry server could not find the specified principal.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_id_gen_group(3sec)**, **sec_id_parse_group(3sec)**, **sec_id_parse_name(3sec)**, **sec_intro(3sec)**.

sec_id_parse_group

Purpose

Translates a global name into group and cell names and UUIDs

Synopsis

```
#include <dce/secidmap.h>

void sec_id_parse_group(
        sec_rgy_handle_t context,
        sec_rgy_name_t global_name,
        sec_rgy_name_t cell_namep,
        uuid_t *cell_idp,
        sec_rgy_name_t group_namep,
        uuid_t *group_idp,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

global_name The global (full) name of the group in **sec_rgy_name_t** form.

Output

cell_namep The output name of the group's home cell in **sec_rgy_name_t** form.

cell_idp A pointer to the UUID of the home cell of the group whose name is in question.

group_namep
The local (with respect to the home cell) name of the group in **sec_rgy_name_t** form.

group_idp A pointer to the UUID of the group whose name is in question.

status A pointer to the completion status. On successful completion, the function returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_id_parse_group()** routine translates a global group name into a cell name and a cell-relative group name. It also returns the UUIDs associated with the group and its home cell.

The routine will not produce translations to any name for which a NULL pointer has been supplied.

Files

/usr/include/dce/secidmap.idl
The **idl** file from which **dce/secidmap.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_id_e_name_too_long
The name is too long for current implementation.

sec_id_e_bad_cell_uuid
The cell UUID is not valid.

sec_rgy_object_not_found
The registry server could not find the specified group.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_id_gen_group(3sec)**, **sec_id_gen_name(3sec)**, **sec_id_parse_group(3sec)**, **sec_id_parse_name(3sec)**, **sec_intro(3sec)**.

sec_id_parse_name

Purpose

Translates a global name into principal and cell names and UUIDs

Synopsis

```
#include <dce/secidmap.h>

void sec_id_parse_name(
        sec_rgy_handle_t context,
        sec_rgy_name_t global_name,
        sec_rgy_name_t cell_namep,
        uuid_t *cell_idp,
        sec_rgy_name_t princ_namep,
        uuid_t *princ_idp,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

global_name The global (full) name of the principal in **sec_rgy_name_t** form.

Output

cell_namep The output name of the principal's home cell in **sec_rgy_name_t** form.

cell_idp A pointer to the UUID of the home cell of the principal whose name is in question.

princ_namep
The local (with respect to the home cell) name of the principal in **sec_rgy_name_t** form.

princ_idp A pointer to the UUID of the principal whose name is in question.

status A pointer to the completion status. On successful completion, the function returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_id_parse_name()** routine translates a global principal name into a cell name and a cell-relative principal name. It also returns the UUIDs associated with the principal and its home cell.

The routine will not produce translations to any name for which a NULL pointer has been supplied.

Permissions Required

Only if *princ_idp* is requested as output does the **sec_id_parse_name()** routine require a permission. In this case, the routine requires at least one permission of any kind on the account whose global principal name is to be translated.

Files

/usr/include/dce/secidmap.idl
The **idl** file from which **dce/secidmap.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_id_e_name_too_long
The name is too long for current implementation.

sec_id_e_bad_cell_uuid
The cell UUID is not valid.

sec_rgy_object_not_found
The registry server could not find the specified principal.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_id_gen_name(3sec)**, **sec_intro(3sec)**.

sec_key_mgmt_change_key

Purpose

Changes a principal's key

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_change_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        unsigned32 key_vno,
        void *keydata,
        sec_timeval_period_t *garbage_collect_time,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** function.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key is to be changed.

key_vno The version number of the new key. If 0 (zero) is specified, the routine will select the next appropriate key version number.

keydata A pointer to a structure of type **sec_passwd_rec_t**.

Output

garbage_collect_time
The number of seconds that must elapse before all currently valid tickets (which are encoded with the current or previous keys) expire. At that time, all obsolete keys may be ‘‘garbage collected,’’ since no valid tickets encoded with those keys will remain outstanding on the network.

status
A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_change_key()** routine performs all activities necessary to update a principal’s key to the specified value. This includes updating any local storage for the principal’s key and also performing any remote operations needed to keep the authentication protocol (or network registry) current. Old keys for the principal are garbage collected if appropriate.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Any error condition will leave the key state unchanged.

sec_key_mgmt_e_key_unavailable
The old key is not present and therefore cannot be used to set a client side authentication context.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_auth_unavailable
The authentication protocol is not available to update the network database or to obtain the necessary network credentials.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_key_mgmt_e_key_unsupported
The key type is not supported.

sec_key_mgmt_e_key_version_ex
A key with this version number already exists.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

sec_rgy_object_not_found
No principal was found with the given name.

sec_login_s_no_memory
A memory allocation error occurred.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_gen_rand_key(3sec)**, **sec_key_mgmt_set_key(3sec)**.

sec_key_mgmt_delete_key

Purpose

Deletes a key from the local storage

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_delete_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        unsigned32 key_vno,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** function.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key is to be deleted.

key_vno The version number of the desired key.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_delete_key()** routine deletes the specified key from the local key store. If an administrator ever discovers or suspects that the security of a server's key has been compromised, the administrator should delete the key immediately with **sec_key_mgmt_delete_key()**. This routine removes the key from the local key storage, which invalidates all extant tickets encoded with the key. If the compromised key is the current one, the principal should change the key with **sec_key_mgmt_change_key()** before deleting it. It is not an error for a process to delete the current key (as long as it is done *after* the network context has been established), but it may seriously inconvenience legitimate clients of a service.

This routine deletes all key types that have the specified key version number. A key type identifies the data encryption algorithm being used (for example, DES). This routine differs from **sec_key_mgmt_delete_key_type()** in that **sec_key_mgmt_delete_key_type()** deletes only the specified key version of the specified key type from the local key store.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Any error condition will leave the key state unchanged.

sec_key_mgmt_e_key_unavailable
The requested key is not present.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_delete_key_type(3sec)**, **sec_key_mgmt_garbage_collect(3sec)**.

sec_key_mgmt_delete_key_type

Purpose

Deletes a key version of a key type from the local key storage

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_delete_key_type(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        void *keytype,
        unsigned32 key_vno,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key type is to be deleted.

keytype A pointer to a value of type **sec_passwd_type_t**. The value identifies the data encryption algorithm that is being used (for example, DES).

key_vno The version number of the desired key.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_delete_key_type()** routine deletes the specified key version of the specified key type from the local key store. It differs from **sec_key_mgmt_delete_key()** in that **sec_key_mgmt_delete_key()** deletes all key types that have the same key version number.

This routine removes the key from the local key storage, which invalidates all extant tickets encoded with the key. If the key in question is the current one, the principal should change the key with **sec_key_mgmt_change_key()** before deleting it. It is not an error for a process to delete the current key (as long as it is done *after* the network context has been established), but it may seriously inconvenience legitimate clients of a service.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

Any error condition will leave the key state unchanged.

sec_key_mgmt_e_key_unavailable
The requested key is not present.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_delete_key(3sec)**, **sec_key_mgmt_garbage_collect(3sec)**.

sec_key_mgmt_free_key

Purpose

Frees the memory used by a key value

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_free_key(
        void *keydata,
        error_status_t *status);
```

Parameters

Input

keydata A pointer to a structure of type **sec_passwd_rec_t**.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**.

Description

The **sec_key_mgmt_free_key()** routine releases any storage allocated for the indicated key data by **sec_key_mgmt_get_key()**. The storage for the key data returned by **sec_key_mgmt_get_key()** is dynamically allocated.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_get_key(3sec)**.

sec_key_mgmt_garbage_collect

Purpose

Deletes obsolete keys

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_garbage_collect(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key information is to be garbage collected.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_garbage_collect()** routine discards any obsolete key information for this principal. An obsolete key is one that can only decode invalid tickets. As an example, consider a key that was in use on Monday, and was only used to encode tickets whose maximum lifetime was 1 day. If that key was changed at 8:00 a.m. Tuesday morning, then it would become obsolete by 8:00 a.m. Wednesday morning, at which time there could be no valid tickets outstanding.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_key_mgmt_e_key_unavailable
Requested key not present.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

sec_rgy_object_not_found
No principal was found with the given name.

sec_login_s_no_memory
A memory allocation error occurred.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_delete_key(3sec)**.

sec_key_mgmt_gen_rand_key

Purpose

Generates a new random key of a specified key type

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_gen_rand_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        void *keytype,
        unsigned32 key_vno,
        void **keydata,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal for whom the key is to be generated.

keytype A pointer to a value of type **sec_passwd_type_t**. The value identifies the data encryption algorithm to be used for the key (for example, DES).

key_vno The version number of the new key.

Output

keydata A pointer to a value of **sec_passwd_rec_t**. The storage for *keydata* is allocated dynamically, so the returned pointer actually indicates a pointer to the key value. The storage for this data may be freed with the **sec_key_mgmt_free_key()** function.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_gen_rand_key()** routine generates a new random key for a specified principal and of a specified key type. The generated key can be used with the **sec_key_mgmt_change_key()** and **sec_key_mgmt_set_key()** routines.

Note that to initialize the random keyseed, the process must first make an authenticated call such as **sec_rgy_site_open()**.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_not_implemented
The specified key type is not supported.

sec_s_no_key_seed
No random key seed has been set.

sec_s_no_memory
Unable to allocate memory.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_change_key(3sec)**, **sec_key_mgmt_set_key(3sec)**.

sec_key_mgmt_get_key

Purpose

Retrieves a key from local storage

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_get_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        unsigned32 key_vno,
        void **keydata,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal to whom the key belongs.

key_vno The version number of the desired key. To return the latest version of the key, set this parameter to **sec_c_key_version_none**.

Output

keydata A pointer to a value of type **sec_passwd_rec_t**. The storage for *keydata* is allocated dynamically, so the returned pointer actually indicates a pointer to the key value. The storage for this data may be freed with the **sec_key_mgmt_free_key()** routine.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_get_key()** routine extracts the specified key from the local key store.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_key_unavailable
The requested key is not present.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_s_no_memory
Unable to allocate memory.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_key_mgmt_get_next_key

Purpose

Retrieves successive keys from the local key storage

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_get_next_key(
        void *cursor,
        idl_char **principal_name,
        unsigned32 *key_vno,
        void **keydata,
        error_status_t *status);
```

Parameters

Input

cursor A pointer to the current cursor position in the local key storage. The cursor position is set via the routine **sec_key_mgmt_initialize_cursor()**.

Output

principal_name
A pointer to a character string indicating the name of the principal associated with the extracted key. Free the storage for the principal name with the **free()** function.

key_vno The version number of the extracted key.

keydata A pointer to a value of type **sec_passwd_rec_t**. The storage for *keydata* is allocated dynamically, so the returned pointer actually indicates a pointer to the key value. The storage for this data may be freed with the **sec_key_mgmt_free_key()** function.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_get_next_key()** routine extracts the key pointed to by the cursor in the local key store and updates the cursor to point to the next key. By repeatedly calling this routine you can scan all the keys in the local store.

Files

/usr/lib/dce/keymgmt.idl
: The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_key_unavailable
: The requested key is not present.

sec_key_mgmt_e_unauthorized
: The caller is not authorized to perform the operation.

sec_s_no_memory
: Unable to allocate memory.

error_status_ok
: The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_get_key(3sec)**, **sec_key_mgmt_initialize_cursor(3sec)**.

sec_key_mgmt_get_next_kvno

Purpose

Retrieves the next eligible key version number for a key

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_get_next_kvno(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        void *keytype,
        unsigned32 *key_vno,
        unsigned32 *next_key_vno,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal associated with the key.

keytype A pointer to a value of type **sec_passwd_type_t**. The value identifies the data encryption algorithm (for example, DES) being used for the key.

Output

key_vno The current version number of the key. Specify NULL if you do not need this value to be returned.

next_key_vno
The next eligible version number for the key. Specify NULL if you do not need this value to be returned.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_get_next_kvno()** routine returns the current and next eligible version numbers for a key from the registry server (not from the local key table). The key is identified via its associated authentication protocol, principal name, and key type. The *arg* value associated with the key is also specified.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_key_unavailable
The requested key is not present.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

sec_rgy_object_not_found
No principal was found with the given name.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_key_mgmt_initialize_cursor

Purpose

Repositions the cursor in the local key store

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_initialize_cursor(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        void *keytype,
        void **cursor,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and

the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key is to be accessed. To access all keys in the local key store, supply NULL for this parameter.

keytype A pointer to the data encryption algorithm (for example, DES) being used for the key.

Output

cursor The returned cursor value. The storage for the cursor information is allocated dynamically, so the returned pointer actually indicates a pointer to the cursor value. The storage for this data may be freed with the **sec_key_mgmt_release_cursor()** routine.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_initialize_cursor()** routine resets the cursor in the local key store.

Use this routine to reposition the cursor before performing a scan of the local store via **sec_key_mgmt_get_next_key()**. The returned cursor value is supplied as input to **sec_key_mgmt_get_next_key()**.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_s_no_memory
Unable to allocate memory.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_get_next_key(3sec)**, **sec_key_mgmt_release_cursor(3sec)**.

sec_key_mgmt_manage_key

Purpose

Automatically changes a principal's key before it expires

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_manage_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. You must prepend the file's absolute filename with **FILE:** and the file must have been created with the **rgy_edit ktadd** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal whose key is to be managed.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_manage_key()** routine changes the specified principal's key on a regular basis, as determined by the local cell's policy. It will run indefinitely, never returning during normal operation, and therefore should be invoked only from a thread that has been devoted to managing keys.

This routine queries the DCE registry to determine the password expiration policy that applies to the named principal. It then idles until a short time before the current key is due to expire and then uses the **sec_key_mgmt_gen_rand_key()** to produce a new random key, updating both the local key store and the DCE registry. This routine also invokes **sec_key_mgmt_garbage_collect()** as needed.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_key_unavailable
The old key is not present and therefore cannot be used to set a client side authentication context.

sec_key_mgmt_e_key_unsupported
The key type is not supported.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

sec_rgy_object_not_found
No principal was found with the given name.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_gen_rand_key(3sec)**, **sec_key_mgmt_garbage_collect(3sec)**.

sec_key_mgmt_release_cursor

Purpose

Releases the memory used by an initialized cursor value

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_release_cursor(
        void **cursor,
        error_status_t *status);
```

Parameters

Input

cursor A pointer to the cursor value for which the storage is to be released.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**.

Description

The **sec_key_mgmt_release_cursor()** routine releases any storage allocated for the indicated cursor value by **sec_key_mgmt_initialize_cursor()**. The storage for the cursor value returned by **sec_key_mgmt_initialize_cursor()** is dynamically allocated.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_initialize_cursor(3sec)**.

sec_key_mgmt_set_key

Purpose

Inserts a key value into the local storage

Synopsis

```
#include <dce/keymgmt.h>

void sec_key_mgmt_set_key(
        sec_key_mgmt_authn_service authn_service,
        void *arg,
        idl_char *principal_name,
        unsigned32 key_vno,
        void *keydata,
        error_status_t *status);
```

Parameters

Input

authn_service

Identifies the authentication protocol using this key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local key file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default key file (**/krb/v5srvtab**) should be used. A key filename specifies that file should be used as the key file. The filename must begin with **FILE:**. If the filename does not begin with **FILE:**, the code will add it.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

principal_name
A pointer to a character string indicating the name of the principal associated with the key to be set.

key_vno The version number of the key to be set.

keydata A pointer to the key value to be set.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_key_mgmt_set_key()** routine performs all local activities necessary to update a principal's key to the specified value. This routine will not update the authentication protocol's value for the principal's key.

In some circumstances, a server may only wish to change its key in the local key storage, and not in the DCE registry. For example, a database system may have several replicas of a master database, managed by servers running on independent machines. Since these servers together represent only one service, they should all share the same key. This way, a user with a ticket to use the database can choose whichever server is least busy. To change the database key, the master server would signal all the replica (slave) servers to change the current key in their local key storage. They would use the **sec_key_mgmt_set_key()** routine, which does not communicate with the DCE registry. Once all the slaves have complied, the master server can then change the registry key and its own local storage.

Files

/usr/include/dce/keymgmt.idl
The **idl** file from which **dce/keymgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_key_mgmt_e_key_unavailable
The old key is not present and therefore cannot be used to set a client side authentication context.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_key_mgmt_e_key_unsupported
The key type is not supported.

sec_key_mgmt_e_key_version_ex
A key with this version number already exists.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_key_mgmt_change_key(3sec)**, **sec_key_mgmt_gen_rand_key(3sec)**.

sec_login_become_delegate

Purpose

Causes an intermediate server to become a delegate in traced delegation chain

Synopsis

```
#include <dce/sec_login.h>

sec_login_handle_t sec_login_become_delegate(
        rpc_authz_cred_handle_t callers_identity,
        sec_login_handle_t my_login_context,
        sec_id_delegation_type_t delegation_type_permitted,
        sec_id_restriction_set_t *delegate_restrictions,
        sec_id_restriction_set_t *target_restrictions,
        sec_id_opt_req_t *optional_restrictions,
        sec_id_opt_req_t *required_restrictions,
        sec_id_compatibility_mode_t compatibility_mode,
        error_status_t *status);
```

Parameters

Input

callers_identity

A handle of type **rpc_authz_cred_handle_t** to the authenticated identity of the previous delegate in the delegation chain. The handle is supplied by the **rpc_binding_inq_auth_caller()** call.

my_login_context

A value of **sec_login_handle_t** that provides an opaque handle to the identity of the client that is becoming the intermediate delegate. The **sec_login_handle_t** that specifies the client's identity is supplied as output of the following calls:

- **sec_login_get_current_context()**, if the client inherited the identity of the current context

- The **sec_login_setup_identity()** and the **sec_login_validate_identity()** pair that together establish an authenticated identity if a new identity was established

Note that this identity specified by **sec_login_handle_t** must be a simple login context; it cannot be a compound identity created by a previous **sec_login_become_delegate()** call.

delegation_type_permitted

A value of **sec_id_delegation_type_t** that specifies the type of delegation to be enabled. The types available are as follows:

sec_id_deleg_type_none
No delegation.

sec_id_deleg_type_traced
Traced delegation.

sec_id_deleg_type_impersonation
Simple (impersonation) delegation.

Note that the initiating client sets the type of delegation. If it is set as traced, all delegates must also specify traced delegation; they cannot specify simple delegation. The same is true if the initiating client sets the delegation type as simple; all subsequent delegates must also specify simple delegation. The intermediate delegates can, however, specify no delegation to indicate that the delegation chain can proceed no further.

delegate_restrictions

A pointer to a **sec_id_restriction_set_t** that supplies a list of servers that can act as delegates for the intermediate client identified by *my_login_context*. These servers are added to delegates permitted by the *delegate_restrictions* parameter of the **sec_login_become_initiator** call.

target_restrictions

A pointer to a **sec_id_restriction_set_t** that supplies a list of servers that can act as targets for the intermediate client identified by *my_login_context*. These servers are added to targets specified by the *target_restrictions* parameter of the **sec_login_become_initiator** call.

optional_restrictions

A pointer to a **sec_id_opt_req_t** that supplies a list of application-defined optional restrictions that apply to the intermediate client

identified by *my_login_context*. These restrictions are added to the restrictions identified by the *optional_restrictions* parameter of the **sec_login_become_initiator** call.

required_restrictions
A pointer to a **sec_id_opt_req_t** that supplies a list of application-defined required restrictions that apply to the intermediate client identified by *my_login_context*. These restrictions are added to the restrictions identified *required_restrictions* parameter of the **sec_login_become_initiator** call.

compatibility_mode
A value of **sec_id_compatibility_mode_t** that specifies the compatibility mode to be used when the intermediate client operates on pre-1.1 servers. The modes available are as follows:

sec_id_compat_mode_none
Compatibility mode is off.

sec_id_compat_mode_initiator
Compatibility mode is on. The pre-1.1 PAC data is extracted from the EPAC of the initiating client.

sec_id_compat_mode_caller
Compatibility mode is on. The pre-1.1 PAC data extracted from the EPAC of the last client in the delegation chain.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_become_delegate()** is used by intermediate servers to become a delegate for the client identified by *callers_identity*. The routine returns a new login context (of type **sec_login_handle_t**) that carries delegation information. This information includes the delegation type, delegate and target restrictions, and any application-defined optional and required restrictions.

The new login context created by this call can then used to to set up authenticated rpc with an intermediate or target server using the **rpc_binding_set_auth_info()** call.

Any delegate, target, required, or optional restrictions specified in this call are added to the restrictions specified by the initiating client and any intermediate clients.

The **sec_login_become_delegate()** call is run only if the initiating client enabled traced delegation by setting the *delegation_type_permitted* parameter in the **sec_login_become_initiator** call to **sec_id_deleg_type_traced**.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

err_sec_login_invalid_delegate_restriction

err_sec_login_invalid_target_restriction

err_sec_login_invalid_opt_restriction

err_sec_login_invalid_req_restriction

sec_login_s_invalid_context

sec_login_s_compound_delegate

sec_login_s_invalid_deleg_type

sec_login_s_invalid_compat_mode

sec_login_s_deleg_not_enabled

error_status_ok

Related Information

Functions: **rpc_binding_inq_auth_caller(3rpc)**, **sec_intro(3sec)**, **sec_login_become_impersonator(3sec)**, **sec_login_become_initiator(3sec)**, **sec_login_get_current_context(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity()**.

sec_login_become_impersonator

Purpose

Used by a server to create a login context and assoicated handle that impersonates the identity of a caller

Synopsis

```
#include <dce/sec_login.h>

sec_login_handle_t sec_login_become_impersonator(
        rpc_authz_cred_handle_t callers_identity,
        sec_login_handle_t my_login_context,
        sec_id_delegation_type_t delegation_type_permitted,
        sec_id_restriction_set_t *delegate_restrictions,
        sec_id_restriction_set_t *target_restrictions,
        sec_id_opt_req_t *optional_restrictions,
        sec_id_opt_req_t *required_restrictions,
        error_status_t *status);
```

Description

The **sec_login_become_impersonator()** is used by intermediate servers to become an impersonator for the client identified by *callers_identity*. The routine returns a new login context (of type **sec_login_handle_t**) that carries delegation information. This information includes the delegation type, delegate, and target restrictions, and any application-defined optional and required restrictions.

The new login context created by this call can then used to to set up authenticated rpc with an intermediate or target server using the **rpc_binding_set_auth_info()** call.

The effective optional and required restrictions are the union of the optional and required restrictions specified in this call and specified by the initiating client and any intermediate clients. The effective target and delegate restrictions are the intersection of the target and delegate restrictions specified in this call and specified by the initiating client and any intermediate clients.

The **sec_login_become_impersonator** call is run only if the initiating client enabled simple delegation by setting the *delegation_type_permitted* parameter in the **sec_login_become_initiator** call to **sec_id_deleg_type_simple**.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

err_sec_login_invalid_delegate_restriction

err_sec_login_invalid_target_restriction

err_sec_login_invalid_opt_restriction

err_sec_login_invalid_req_restriction

sec_login_s_invalid_deleg_type

sec_login_s_invalid_compat_mode

sec_login_s_deleg_not_enabled

error_status_ok

Related Information

Functions: **rpc_binding_inq_auth_caller(3rpc)**, **sec_intro(3sec)**, **sec_login_become_initiator(3sec)**.

sec_login_become_initiator

Purpose

Constructs a new login context that enables delegation for the calling client

Synopsis

```
#include <dce/sec_login.h>

sec_login_handle_t sec_login_become_initiator(
        sec_login_handle_t my_login_context,
        sec_id_delegation_type_t delegation_type_permitted,
        sec_id_restriction_set_t *delegate_restrictions,
        sec_id_restriction_set_t *target_restrictions,
        sec_id_opt_req_t *optional_restrictions,
        sec_id_opt_req_t *required_restrictions,
        sec_id_compatibility_mode_t compatibility_mode,
        error_status_t *status);
```

Parameters

Input

my_login_context

A value of **sec_login_handle_t** that provides an opaque handle to the identity of the client that is enabling delegation. The **sec_login_handle_t** that specifies the client's identity is supplied as output of the following calls:

- **sec_login_get_current_context()** if the client inherited the identity of the current context
- The **sec_login_setup_identity()** and the **sec_login_validate_identity()** pair that together establish an authentiated identity if a new identity was established

delegation_type_permitted

A value of **sec_id_delegation_type_t** that specifies the type of delegation to be enabled. The types available are as follows:

sec_id_deleg_type_none
No delegation.

sec_id_deleg_type_traced
Traced delegation.

sec_id_deleg_type_impersonation
Simple (impersonation) delegation.

Note each subsequent intermediate delegate of the delegation chain started by the initiating client must set the delegation type to traced if the initiating client set it to traced or to simple if the initiating client set it to simple. Intermediate delegates, however, can set the delegation type to no delegation to indicate that the delegation chain can proceed no further.

delegate_restrictions
A pointer to a **sec_id_restriction_set_t** that supplies a list of servers that can act as delegates for the client initiating delegation.

target_restrictions
A pointer to a **sec_id_restriction_set_t** that supplies a list of servers that can act as targets for the client initiating delegation.

optional_restrictions
A pointer to a **sec_id_opt_req_t** that supplies a list of application-defined optional restrictions that apply to the client initiating delegation.

required_restrictions
A pointer to a **sec_id_opt_req_t** that supplies a list of application-defined required restrictions that apply to the client initiating delegation.

compatibility_mode
A value of **sec_id_compatibility_mode_t** that specifies the compatibility mode to be used when the initiating client interacts with pre-1.1 servers. The modes available are as follows:

sec_id_compat_mode_none
Compatibility mode is off.

sec_id_compat_mode_initiator
Compatibility mode is on. The pre-1.1 PAC data is extracted from the EPAC of the initiating client.

sec_id_compat_mode_caller
Compatibility mode is on. The pre-1.1 PAC data extracted from the EPAC of the last client in the delegation chain.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_become_initiator()** enables delegation for the calling client by constructing a new login context (in a **sec_login_handle_t**) that carries delegation information. This information includes the delegation type, delegate, and target restrictions, and any application-defined optional and required restrictions.

The new login context is then used to to set up authenticated rpc with an intermediate server using the **rpc_binding_set_auth_info()** call. The intermediary can continue the delegation chain by calling **sec_login_become_delegate** (if the delegation type is **sec_id_deleg_type_traced**) or **sec_login_become_impersonator** (if the delegation type is **sec_id_deleg_type_impersonation**).

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

err_sec_login_invalid_delegate_restriction

err_sec_login_invalid_target_restriction

err_sec_login_invalid_opt_restriction

err_sec_login_invalid_req_restriction

error_status_ok

sec_login_s_invalid_compat_mode

sec_login_s_invalid_context

sec_login_s_invalid_deleg_type

Related Information

Functions: **sec_intro(3sec)**, **sec_login_become_delegate(3sec)**, **sec_login_become_impersonator(3sec)**, **sec_login_get_current_context(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity()**.

sec_login_certify_identity

Purpose

Certifies the network authentication service

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_certify_identity(
        sec_login_handle_t login_context,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_certify_identity()** routine certifies that the security server used to set up and validate a login context is legitimate. A legitimate server is one that knows the host machine's secret key. On some systems, this may be a privileged operation.

Information may be retrieved via **sec_login_get_pwent()**, **sec_login_get_groups()**, and **sec_login_get_expiration()** from an uncertified login context, but such information cannot be trusted. All system login programs that use the **sec_login** interface must call **sec_login_certify_identity()** to certify the security server. If

they do not, they open the local file system to attacks by imposter Security servers returning suspect local process credentials (UUID and group IDs). This operation updates the local registry with the login context credentials if the certification check succeeds.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Return Values

The routine returns a **boolean32** value that is TRUE if the certification was successful, and FALSE otherwise.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_config
The DCE configuration (**dce_config**) information is not available.

sec_login_s_context_invalid
The input context is invalid.

sec_login_s_default_use
It is an error to try to certify the default context.

error_status_ok
The call was successful.

Examples

Applications wishing to perform a straightforward login can use the **sec_login** package as follows:

```
if
(sec_login_setup_identity(user_name, sec_login_no_flags, &login_context,
                          &st)) {
   ... get password from user...

   if (sec_login_validate_identity(login_context, password,
                          &reset_passwd, &auth_src, &st)) {

      if (!sec_login_certify_identity(login_context, &st))
          exit(error_weird_auth_svc);

      sec_login_set_context(login_context, &st);

      if (auth_src != sec_login_auth_src_network)
         printf("no network credentials");

      if (reset_passwd) {
         ... get new password from user, reset registry record ...
      };

      sec_login_get_pwent(login_context, &pw_entry, &st);

      if (pw_entry.pw_expire < todays_date) {
         sec_login_purge_context(&login_context, &st);
         exit(0)
      }

      ... any other application specific login valid actions ...
      }

   } else {
      sec_login_purge_context(&login_context, &st);

      ... application specific login failure actions ...
   }
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_get_expiration(3sec)**, **sec_login_get_groups(3sec)**, **sec_login_get_pwent(3sec)**.

sec_login_cred_get_delegate

Purpose Returns a handle to the privilege attributes of an intermediary in a delegation chain. Used by clients.

Synopsis

```
#include <dce/sec_login.h>

sec_cred_pa_handle_t sec_login_cred_get_delegate(
        sec_login_handle_t login_context,
        sec_cred_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input

login_context
: A value of **sec_login_handle_t** that provides an opaque handle to a login context for which delegation has been enabled. The **sec_login_handle_t** that specifies the identity is supplied as output of the **sec_login_become_delegate()** call.

Input/Output

cursor
: As input, a pointer to a cursor of type **sec_cred_cursor_t** that has been initialized by the **sec_login_cred_init_cursor()** call. As an output parameter, *cursor* is a pointer to a cursor of type **sec_cred_cursor_t** that is positioned past the principal whose privilege attributes have been returned in this call.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_cred_get_delegate()** routine returns a handle of type **sec_login_handle_t** to the the privilege attributes of an intermediary in a delegation chain that performed an authenticated RPC operation.

This call is used by clients. Servers use the **sec_cred_get_delegate()** routine to return the privilege attribute handle of an intermediary in a delegation chain.

The login context identified by *login_context* contains all members in the delegation chain. This call returns a handle (**sec_cred_pa_handle_t**) to the privilege attributes of one of the delegates in the login context. The **sec_cred_pa_handle_t** returned by this call is used in other **sec_cred_get_*** calls to obtain privilege attribute information for a single delegate.

To obtain the privilege attributes of each delegate in the credential handle identified by *callers_identity*, execute this call until the message **sec_cred_s_no_more_entries** is returned.

Before you execute **sec_login_cred_get_delegate()**, you must execute a **sec_login_cred_init_cursor()** call to initialize a cursor of type **sec_cred_cursor_t**.

Use the **sec_cred_free_pa_handle() sec_cred_free_cursor()** calls to free the resources allocated to the **sec_cred_pa_handle_t** and *cursor*.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_cursor

sec_cred_s_no_more_entries

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_cred_get_deleg_restrictions(3sec)**, **sec_cred_get_delegation_type(3sec)**, **sec_cred_get_extended_attrs(3sec)**, **sec_cred_get_opt_restrictions(3sec)**, **sec_cred_get_pa_date(3sec)**, **sec_cred_get_req_restrictions(3sec)**, **sec_cred_get_tgt_restrictions(3sec)**, **sec_cred_get_v1_pac(3sec)**, **sec_login_cred_init_cursor(3sec)**.

sec_login_cred_get_initiator

Purpose

Returns information about the delegation initiator in a specified login context

Synopsis

```
#include <dce/sec_login.h>

sec_cred_pa_handle_t sec_login_cred_get_initiator(
        sec_login_handle_t login_context,
        error_status_t *status);
```

Parameters

Input

login_context
: A value of **sec_login_handle_t** that provides an opaque handle to a login context for which delegation has been enabled.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_cred_get_initiator()** routine returns a handle of type **sec_cred_pa_handle_t** to the privilege attributes of the delegation initiator.

The login context identified by *login_context* contains all members in the delegation chain. This call returns a handle (**sec_cred_pa_handle_t**) to the privilege attributes of the initiator. The **sec_cred_pa_handle_t** returned by this call is used in other **sec_cred_get_*** calls to obtain privilege attribute information for the initiator single delegate.

Use the **sec_cred_free_pa_handle()** call to free the resources allocated to the **sec_cred_pa_handle_t** handle.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_invalid_context

error_status_ok

Related Information

Functions: **sec_cred_get_deleg_restrictions(3sec)**, **sec_cred_get_delegation_type(3sec)**, **sec_cred_get_extended_attrs(3sec)**, **sec_cred_get_opt_restrictions(3sec)**, **sec_cred_get_pa_date(3sec)**, **sec_cred_get_req_restrictions(3sec)**, **sec_cred_get_tgt_restrictions(3sec)**, **sec_cred_get_v1_pac(3sec)**, **sec_intro(3sec)**.

sec_login_cred_init_cursor

Purpose

Initializes a **sec_cred_cursor_t**

Synopsis

```
#include <dce/sec_cred.h>

void sec_login_cred_init_cursor (
        sec_cred_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor As input, a pointer to a **sec_cred_cursor_t** to be initialized. As output, a pointer to an initialized **sec_cred_cursor_t**.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_cred_init_cursor()** routine allocates and initializes a cursor of type **sec_cursor_t** for use with the **sec_login_cred_get_delegate()** call.

Use the **sec_cred_free_cursor()** call to free the resources allocated to *cursor*.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_cred_s_invalid_cursor

sec_login_s_no_memory

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_login_cred_get_delegate(3sec)**.

sec_login_disable_delegation

Purpose

Disables delegation for a specified login context

Synopsis

```
#include <dce/sec_login.h>

sec_logon_handle_t *sec_login_disable_delegation(
        sec_login_handle_t login_context,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context for which delegation has been enabled.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_disable_delegation()** routine disables delegation for a specified login context. It returns a new login context of type **sec_login_handle_t** without any delegation information, thus preventing any further delegation.

Files

/usr/include/dce/sec_login.idl
: The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_invalid_context

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_login_become_delegate(3sec)**, **sec_login_become_impersonator(3sec)**, **sec_login_become_initiator(3sec)**.

sec_login_export_context

Purpose

Creates an exportable login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_export_context(
        sec_login_handle_t login_context,
        unsigned32 buf_len,
        idl_byte buf[ ],
        unsigned32 *len_used,
        unsigned32 *len_needed,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

buf_len
: An unsigned 32-bit integer containing the allocated length (in bytes) of the buffer that is to contain the login context.

Output

buf[ ]
: An idl_byte array that contains the exportable login context upon return.

len_used
: A pointer to an unsigned 32-bit integer indicating the number of bytes needed for the entire login context, up to *buf_len*.

len_needed
: If the allocated length of the buffer is too short, an error is issued (**sec_login_s_no_memory**), and on return this pointer indicates the number of bytes necessary to contain the login context.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_export_context()** routine obtains an exportable version of the login context information. This information may be passed to another process running on the same machine.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_memory
Not enough space was allocated for the *buf*[ ] array. The *len_needed* parameter will point to the needed length.

sec_login_s_handle_invalid
The login context handle is invalid.

sec_login_s_context_invalid
The login context specified by the input handle is invalid.

Related Information

Functions: **sec_login_import_context(3sec)**, **sec_intro(3sec)**.

sec_login_free_net_info

Purpose

Frees storage allocated for a principal's network information

Synopsis

```
#include <dce/sec_login.h>

void sec_login_free_net_info(
        sec_login_net_info_t *net_info);
```

Parameters

Input/Output

net_info A pointer to the **sec_login_net_info_t** structure to be freed.

Description

The **sec_login_free_net_info()** routine frees any memory allocated for a principal's network information. Network information is returned by a previous successful call to **sec_login_inquire_net_info()**.

Cautions

This routine does not return any completion codes. Make sure that you supply a valid **sec_login_net_info_t** address. The routine simply frees a range of storage beginning at the supplied address, without regard to the actual contents of the storage.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_inquire_net_info(3sec)**.

sec_login_get_current_context

Purpose

Returns a handle to the current login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_get_current_context(
        sec_login_handle_t *login_context,
        error_status_t *status);
```

Parameters

Output

login_context
: A pointer to an opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_get_current_context()** routine retrieves a handle to the login context for the currently established network identity. The context returned is created from locally cached data so subsequent data extraction operations may return some NULL values.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
There was no current context to retrieve. (See **sec_login_setup_identity(3sec)** for information about how to set up, validate, and implement a login context.)

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_get_current_context()** routine as part of a process to change the groupset:

```
sec_login_get_current_context(&login_context, &st);

sec_login_get_groups(login_context, &num_groups, &groups, &st);

   ...the group IDs have to be converted from the returned UNIX
   numbers into UUIDs (use sec_rgy_pgo_unix_num_to_id(3sec)...

for (i=0; i < num_groups; i++) {
   ... query whether the user wants to discard any of the current
   group memberships.  Copy new groupset to the new_groups array ...
}

if ( !sec_login_newgroups(sec_login_no_flags, num_new_groups,
        new_groups, &login_context, &st)) {
   if (st == sec_login_s_groupset_invalid)
```

```
            printf("New groupset invalid);

        ... application specific error handling ...
    }
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_setup_identity(3sec)**.

sec_login_get_expiration

Purpose

Returns the TGT lifetime for an authenticated identity

Synopsis

```
#include <dce/sec_login.h>

void sec_login_get_expiration(
        sec_login_handle_t login_context,
        signed32 *identity_expiration,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

Output

identity_expiration
: The lifetime of the ticket-granting ticket (TGT) belonging to the authenticated identity identified by *login_context*. It can be used in the same ways as a UNIX **time_t**.

status
: A pointer to the completion status.

Description

The **sec_login_get_expiration()** routine extracts the lifetime for the TGT belonging to the authenticated identity contained in the login context. The lifetime value is filled in if available; otherwise, it is set to 0 (zero). This routine allows an application to

tell an interactive user how long the user's network login (and authenticated identity) will last before having to be refreshed.

The routine works only on previously certified contexts.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is invalid.

sec_login_s_default_use
There was illegal use of the default login handle.

sec_login_s_not_certified
The login context has not been certified.

sec_login_s_no_current_context
The calling process has no context of its own.

error_status_ok
The call was successful.

Examples

Since the authenticated network identity for a process has a finite lifetime, there is a risk it will expire during some long network operation, preventing the operation from completing. To avoid this situation, an application might, before initiating a long operation, use the **sec_login** package to check the expiration time of its identity and refresh it if there is not enough time remaining to complete the operation. After refreshing the identity, the process must validate it again with **sec_login_validate_identity()**.

```
sec_login_get_expiration(login_context, &expire_time, &st);

if (expire_time < (current_time + operation_duration)) {

     if (!sec_login_refresh_identity(login_context, &st)) {
      if (st == sec_login_s_refresh_ident_bad) {
         ... identity has changed ...
      } else {
         ... login context cannot be renewed ...
         exit(error_context_not_renewable);
      }

     if (sec_login_validate_identity(login_context, password,
                           &reset_passwd, &auth_src, &st)) {
         ... identity validated ...
      } else {
         ... validation failed ...
         exit(error_validation_failure);
      }
   }
}

operation();
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_get_current_context(3sec)**.

sec_login_get_groups

Purpose

Returns the group set from a login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_get_groups(
        sec_login_handle_t login_context,
        unsigned32 *num_groups,
        signed32 **group_set,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

Output

num_groups
: An unsigned 32-bit integer indicating the total number of groups returned in the *group_set* array.

group_set
: The list of groups to which the user belongs.

status
: A pointer to the completion status.

Description

The **sec_login_get_groups()** routine returns the groups contained in the supplied login context. Part of a network identity is a list of the various groups to which the principal

belongs. The groups are used to determine a user's access to various objects and services. This routine extracts from the login context a list of the groups for which the user has established network privileges.

The routine works only on previously validated contexts.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is not valid.

sec_login_s_info_not_avail
The login context has no UNIX information.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_not_certified
The login context has not been certified.

sec_login_s_not_certified
The login context is not certified.

sec_rgy_object_not_found
The registry server could not find the specified login context data.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_get_groups()** routine as part of a process to change the groupset:

```
sec_login_get_current_context(&login_context, &st);

sec_login_get_groups(login_context, &num_groups, &groups, &st);

   ...the group IDs have to be converted from the returned UNIX
   numbers into UUIDs (use sec_rgy_pgo_unix_num_to_id(3sec)...

for (i=0; i < num_groups; i++) {
   ... query whether the user wants to discard any of the current
   group memberships.  Copy new groupset to the new_groups array ...
}

if ( !sec_login_newgroups(sec_login_no_flags, num_new_groups,
        new_groups, &login_context, &st)) {
   if (st == sec_login_s_groupset_invalid)
        printf("New groupset invalid);

   ... application specific error handling ...
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_get_projlist(3sec)**.

sec_login_get_pwent

Purpose

Returns a **passwd**-style entry for a login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_get_pwent(
        sec_login_handle_t login_context,
        sec_login_passwd_t *pwent,
        error_status_t *status);
```

Parameters

Input

login_context

An opaque handle to login context data. The login context contains, among other data, the account principal name and Universal Unique Identifier (UUID), account restrictions, records of group membership, and the process home directory. (See the **sec_intro(3sec)** reference page for more details about the login context.)

Output

pwent

A pointer to a pointer to the returned **passwd**-style structure. The particular structure depends on the underlying system. For example, on a system with a **passwd** structure like that supported by 4.4BSD and OSF/1, the structure (found in **/usr/include/pwd.h**) is as follows:

```
struct passwd {
    char   *pw_name;     /* user name */
    char   *pw_passwd;   /* encrypted password */
    int     pw_uid;      /* user uid */
    int     pw_gid;      /* user gid */
```

```
    time_t  pw_change;   /* password change time */
    char   *pw_class;    /* user access class */
    char   *pw_gecos;    /* miscellaneous account info */
    char   *pw_dir;      /* home directory */
    char   *pw_shell;    /* default shell */
    time_t  pw_expire;   /* account expiration */
};
```

status A pointer to the completion status. On successful completion, the routine returns one of the following status codes:

error_status_ok
Indicates that the login context has been validated and certified.

sec_login_s_not_certified
Indicates that the login context has been validated, but not certified. Although this code indicates successful completion, it warns you that the context is not validated.

If the call does not complete successfully, it returns an error.

Description

The **sec_login_get_pwent()** routine creates a **passwd**-style structure for the current network login context. This is generally useful for establishing the local operating system context. Applications that require all of the data normally extracted via **getpwnam()** should extract that data from the login context with this call.

This routine works only on explicitly created (not inherited or imported) contexts.

Caution: The returned **sec_login_passwd_t** structure points to data stored in the structure indicated by the *login_context* pointer, and must be treated as read-only data. Writing to these data objects may cause unexpected failures.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is invalid.

sec_login_s_not_certified
The login context has not been certified.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_info_not_avail
The login context has no UNIX information.

sec_rgy_object_not_found
The registry server could not find the specified login context data.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_get_pwent()** routine:

```
#include <pwd.h>
...
struct passwd *pwd;
...
sec_login_get_pwent(login_context,(sec_login_passwd_t*)&pwd,&status);
...
printf ("%s",pwd->pw_name);
```

Related Information

Functions: **sec_intro(3sec)**.

sec_login_import_context

Purpose

Imports a login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_import_context(
        unsigned32 buf_len,
        idl_byte buf[ ],
        sec_login_handle_t *login_context,
        error_status_t *status);
```

Parameters

Input

buf_len The allocated length (in bytes) of the buffer containing the login context.

buf[ ] An idl_byte array containing the importable login context.

Output

login_context
An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_import_context()** routine imports a context obtained via a call to **sec_login_export_context()** performed on the same machine. To import a login

context, users must have the appropriate privileges. Non-privileged users can import only their own login context; privileged users can import the login contexts created by any users.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is not valid.

sec_login_s_default_use
Illegal use of the default login handle occurred.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_export_context(3sec)**.

sec_login_init_first

Purpose

Initializes the default context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_init_first(
        error_status_t *status);
```

Parameters

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_init_first()** routine initializes the default context inheritance mechanism. If the default inheritance mechanism is already initialized, the operation fails. Typically, this routine is called by the initial process at machine boot time to initialize the default context inheritance mechanism for the host machine process hierarchy.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_default_use
The default context is already initialized.

sec_login_s_privileged
An unprivileged process was called in.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_setup_first(3sec)**, **sec_login_validate_first(3sec)**.

sec_login_inquire_net_info

Purpose

Returns a principal's network information

Synopsis

```
#include <dce/sec_login.h>

void sec_login_inquire_net_info(
        sec_login_handle_t login_context,
        sec_login_net_info_t *net_info,
        error_status_t *status);
```

Parameters

Input

login_context

An opaque handle to the login context for the desired principal. (See **sec_intro(3sec)** for more details about the login context.)

Output

net_info

A pointer to the returned **sec_login_net_info_t** data structure that contains the principal's network information. The **sec_login_net_info_t** structure is defined as follows:

```
typedef struct {
   sec_id_pac_t  pac;
   unsigned32    acct_expiration_date;
   unsigned32    passwd_expiration_date;
   unsigned32    identity_expiration_date;
} sec_login_net_info_t;
};
```

status

A pointer to the completion status.

Description

The **sec_login_inquire_net_info()** routine returns network information for the principal identified by the specified login context. The network information consists of the following:

- The privilege attribute certificate (PAC) that describes the identity and group memberships of the principal.
- The expiration date for the principal's account in the DCE registry.
- The expiration date for the principal's password in the DCE registry.
- The lifetime for the principal's authenticated network identity. This is the lifetime of the principal's TGT (see the **sec_login_get_expiration()** routine).

A value of 0 (zero) for an expiration date means there is no expiration date. In other words, the principal's account, password, or authenticated identity is good indefinitely.

To remove the returned *net_info* structure when it is no longer needed, use **sec_login_free_net_info()**.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_not_certified
The login context is not certified.

sec_login_s_context_invalid
The login context is not valid.

sec_login_s_no_current_context
The default context was specified, but none exists.

sec_login_s_auth_local
Operation not valid on local context. The call's identity was not authenticated.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_free_net_info(3sec)**, **sec_login_get_expiration(3sec)**.

sec_login_newgroups

Purpose

Changes the group list for a login context

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_newgroups(
        sec_login_handle_t login_context,
        sec_login_flags_t flags,
        unsigned32 num_local_groups,
        sec_id_t local_groups[ ],
        sec_login_handle_t *restricted_context,
        error_status_t *status);
```

Parameters

Input

login_context

An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

flags

A set of flags of type **sec_login_flags_t**. These contain information about how the new network credentials will be used. Currently, the only flag used is **sec_login_credentials_private**, that, when set, implies that the new context is only to be used by the calling process. If this flag is not set (*flags* = **sec_login_no_flags**), descendants of the calling process may also use the new network credentials.

num_local_groups

An unsigned 32-bit integer containing the number of local group identities to include in the new context.

local_groups[ ]
An array of **sec_id_t** elements. Each element contains the UUID of a local group identity to include in the new context. These identities are local to the cell. Optionally, each element may also contain a pointer to a character string containing the name of the local group.

Output

restricted_context
An opaque handle to the login context containing the changed group list.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_newgroups()** routine changes the group list for the specified login context. Part of a network identity is a list of the various groups to which a principal belongs. The groups are used to determine a user's access to various objects and services. This routine returns a new login context that contains the changed group list. To remove the new login context when it is no longer needed, use **sec_login_purge_context()**.

This operation does not need to be validated as the user identity does not change. Consequently, knowledge of the password is not needed.

Notes

Currently you can have only groups from the local cell.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Return Values

This routine returns TRUE when the new login context is successfully established.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_auth_local
Operation not valid on local context.

sec_login_s_default_use
It is an error to try to certify the default context.

sec_login_s_groupset_invalid
The input list of group names is invalid. There may be groups to which the caller does not belong, or the list may contain groups that do not exist.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_newgroups()** routine as part of a process to change the groupset:

```
sec_login_get_current_context(&login_context, &st);

sec_login_get_groups(login_context, &num_groups, &groups, &st);

   ...the group IDs have to be converted from the returned UNIX
   numbers into UUIDs (use sec_rgy_pgo_unix_num_to_id(3sec)...

for (i=0; i < num_groups; i++) {
   ... query whether the user wants to discard any of the current
   group memberships.  Copy new groupset to the new_groups array ...
}

if ( !sec_login_newgroups(sec_login_no_flags, num_new_groups,
  new_groups, &login_context, &st)) {
   if (st == sec_login_s_groupset_invalid)
  printf("New groupset invalid);
```

```
    ... application specific error handling ...
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_get_groups(3sec)**, **sec_login_purge_context(3sec)**.

sec_login_purge_context

Purpose

Destroys a login context and frees its storage

Synopsis

```
#include <dce/sec_login.h>

void sec_login_purge_context(
        sec_login_handle_t *login_context,
        error_status_t *status);
```

Parameters

Input

login_context

A pointer to an opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.) Note that a pointer to the handle is submitted, so the handle may be reset to NULL upon successful completion.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_purge_context()** routine frees any storage allocated for the specified login context and destroys the associated network credentials, if any exist.

Cautions

Applications must be cautious when purging the current context as this destroys network credentials for all processes that share the credentials.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_context_invalid
The login context itself is not valid.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_purge_context()** routine as part of a straightforward login process:

```
if (sec_login_setup_identity(user_name, sec_login_no_flags,
            &login_context, &st)) {
   ... get password from user...

   if (sec_login_validate_identity(login_context, password,
                           &reset_passwd, &auth_src, &st)) {

      if (!sec_login_certify_identity(login_context, &st))
          exit(error_wierd_auth_svc);
```

```
        sec_login_set_context(login_context, &st);

        if (auth_src != sec_login_auth_src_network)
           printf("no network credentials");

        if (reset_passwd) {
           ... get new password from user, reset registry record ...
        };

        sec_login_get_pwent(login_context, &pw_entry, &st);

        if (pw_entry.pw_expire < todays_date) {
           sec_login_purge_context(&login_context, &st);
           exit(0)
        }

        ... any other application specific login valid actions ...
        }

     } else {
        sec_login_purge_context(&login_context, &st);

        ... application specific login failure actions ...
     }
  }
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_set_context(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_refresh_identity

Purpose

Refreshes an authenticated identity for a login context

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_refresh_identity(
        sec_login_handle_t login_context,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory.

Output

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_refresh_identity()** routine refreshes a previously established identity. It operates on an existing valid context, and cannot be used to change credentials associated with that identity. The refreshed identity reflects changes that affect ticket lifetimes, but not other changes. For example, the identity will reflect a change to maximum ticket lifetime, but not the addition of the identity as a member to a group. Only a DCE login reflects all administrative changes made since the last login.

The refreshed identity must be validated with **sec_login_validate_identity()** before it can be used.

It is an error to refresh a locally authenticated context.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is not valid.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_no_memory
Not enough memory is available to complete the operation.

error_status_ok
The call was successful.

Examples

Since the authenticated network identity for a process has a finite lifetime, there is a risk it will expire during some long network operation, preventing the operation from completing.

For a server application that must run with an authenticated network identity because they themselves sometimes act as clients of another server, the **sec_login** calls can be used to check the network identity expiration date, run **sec_login_refresh_identity** and **sec_login_validate_identity** before the expiration. This will prevent interruptions in the server's operation due to the restrictions in network access applied to an unauthenticated identity.

```
sec_login_get_expiration(login_context, &expire_time, &st);

if (expire_time < (current_time + operation_duration)) {

   if (!sec_login_refresh_identity(login_context, &st)) {
         ... login context cannot be renewed ...
         ... sleep and try again ....
      }

} else {

     if (sec_login_validate_identity(login_context, password,
                           &reset_passwd, &auth_src, &st)) {
         ... identity validated ...
      } else {
         ... validation failed ...
         exit(error_validation_failure);
      }
   }
}

operation();
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_release_context

Purpose

Frees storage allocated for a login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_release_context(
        sec_login_handle_t *login_context,
        error_status_t *status);
```

Parameters

Input/Output

login_context
: A pointer to an opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

Output

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_release_context()** routine frees any memory allocated for a login context. Unlike **sec_login_purge_context()**, it does not destroy the associated network credentials that still reside in the credential cache.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_context_invalid
The login context itself is invalid.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_purge_context(3sec)**.

sec_login_set_context

Purpose

Creates network credentials for a login context

Synopsis

```
#include <dce/sec_login.h>

void sec_login_set_context(
        sec_login_handle_t login_context,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

Output

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_set_context()** routine sets the network credentials to those specified by the login context. This context must have been previously validated. Contexts acquired through **sec_login_get_current_context()** or **sec_login_newgroups()** do not need to be validated since those routines return previously validated contexts.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_context_invalid
The login context itself is invalid.

sec_login_s_default_use
Illegal use of the default login handle occurred.

sec_login_s_auth_local
Operation not valid on local context.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_set_context()** routine as part of a straightforward login process:

```
if (sec_login_setup_identity(user_name, sec_login_no_flags,
                          &login_context, &st)) {
   ... get password from user...

   if (sec_login_validate_identity(login_context, password,
                            &reset_passwd, &auth_src, &st)) {

      if (!sec_login_certify_identity(login_context, &st))
          exit(error_weird_auth_svc);

      sec_login_set_context(login_context, &st);
```

```
            if (auth_src != sec_login_auth_src_network)
               printf("no network credentials");

            if (reset_passwd) {
               ... get new password from user, reset registry record ...
            };

            sec_login_get_pwent(login_context, &pw_entry, &st);

            if (pw_entry.pw_expire < todays_date) {
               sec_login_purge_context(&login_context, &st);
               exit(0)
            }

            ... any other application specific login valid actions ...
            }

         } else {
            sec_login_purge_context(&login_context, &st);

            ... application specific login failure actions ...
         }
      }
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_set_extended_attrs

Purpose

Constructs a new login context that contains extended registry attributes

Synopsis

```
#include <dce/sec_login.h>

sec_login_handle_t sec_login_set_extended_attrs(
        sec_login_handle_t my_login_context,
        unsigned32 num_attributes,
        sec_attr_t attributes[ ],
        error_status_t *status);
```

Parameters

Input

my_login_context
: A value of **sec_login_handle_t** that provides an opaque handle to the identity of the calling client.

num_attributes
: An unsigned 32-bit integer that specifies the number of elements in the *attributes[ ]* array. The number must be greater than 0.

attributes[ ]
: An array of values of type **sec_attr_t** that specifies the list of attributes to be set in the new login context.

Output

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_set_extended_attrs()** constructs a login context that contains extended registry attributes that have been established for the object identified by *my_login_context*. The attributes themselves must have been established and attached to the object using the extended registry attribute API.

The input *attributes[ ]* array of **sec_attr_t** values should specify the *attr_id* field for each requested attribute. Since the lookup is by attribute type ID only, set the *attribute.attr_value.attr_encoding* field to **sec_attr_enc_void** for each attribute. Note that **sec_attr_t** is an extended registry attribute data type. For more information on extended registry attributes, see the description of the **sec_attr** calls in this document and the *DCE 1.2.2 Application Development Guide—Core Components*.

You cannot use this call to add extended registry attributes to a delegation chain. If you pass in a login context that refers to a delegation chain, an invalid context error will be returned.

The routine returns a new login context of type **sec_login_handle_t** that includes the attributes specified in the *attributes[ ]* array.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_invalid_context

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_login_become_impersonator(3sec)**, **sec_login_set_context(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity(3sec)**, **sec_rgy_attr_*(3sec)** calls.

sec_login_setup_first

Purpose

Sets up the default network context

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_setup_first(
        sec_login_handle_t *init_context,
        error_status_t *status);
```

Parameters

Output

init_context
: A pointer to an opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. In this call, the context will be that of the host machine initial process. (See **sec_intro(3sec)** for more details about the login context.)

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_setup_first()** routine sets up the default context network identity. If the default context already contains valid credentials, the routine fails. Typically, this routine is called from the security validation service of the **dced** process to breathe life into the default credentials for the host machine process hierarchy.

This routine uses the host name available via the local **dce_config** interface as the principal name for the setup, so it does need a principal name as input.

Return Values

The routine returns a **boolean32** value that is TRUE if the setup was successful, and FALSE otherwise.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_default_use
The default context is already in use and does not need to be set up again.

sec_login_s_no_current_context
The calling process has no context of its own.

sec_login_s_privileged
An unprivileged process was called in.

sec_login_s_config
The DCE configuration (**dce_config**) information is not available.

sec_rgy_object_not_found
The principal does not exist.

sec_rgy_server_unavailable
The network registry is not available.

sec_login_s_no_memory
A memory allocation error occurred.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_init_first(3sec)**, **sec_login_validate_first(3sec)**.

sec_login_setup_identity

Purpose

Sets up the user's network identity

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_setup_identity(
        unsigned_char_p_t principal,
        sec_login_flags_t flags,
        sec_login_handle_t *login_context,
        error_status_t *status);
```

Parameters

Input

principal A pointer (type **unsigned_char_p_t**) indicating a character string containing the principal name on the registry account corresponding to the calling process.

flags A set of flags of type **sec_login_flags_t**. These contain information about how the new network credentials are to be used.

Output

login_context
A pointer to an opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_setup_identity()** routine creates any local context necessary to perform authenticated network operations. It does not establish any local operating system context; that is the responsibility of the caller. It is the standard network login function. The network identity set up by this operation cannot be used until it is validated via **sec_login_validate_identity()**.

The **sec_login_setup_identity()** operation and the **sec_login_validate_identity()** operation are two halves of a single logical operation. Together they collect the identity data needed to establish an authenticated identity.

Notes

Neither **sec_login_setup_identity()** nor **sec_login_validate_identity()** check for account or identity expiration. The application program using this interface is responsible for such checks.

Return Values

The routine returns TRUE if the identity has been successfully established.

Files

/usr/include/dce/sec_login.idl
: The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
: The principal does not exist.

sec_rgy_server_unavailable
: The network registry is not available.

sec_login_s_no_memory
Not enough memory is available to complete the operation.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_setup_identity()** routine as part of a straightforward login process:

```
if (sec_login_setup_identity(user_name, sec_login_no_flags,
                          &login_context, &st)) {
   ... get password from user...

   if (sec_login_validate_identity(login_context, password,
                            &reset_passwd, &auth_src, &st)) {

      if (!sec_login_certify_identity(login_context, &st))
          exit(error_weird_auth_svc);

      sec_login_set_context(login_context, &st);

      if (auth_src != sec_login_auth_src_network)
         printf("no network credentials");

      if (reset_passwd) {
         ... get new password from user, reset registry record ...
      };

      sec_login_get_pwent(login_context, &pw_entry, &st);

      if (pw_entry.pw_expire < todays_date) {
         sec_login_purge_context(&login_context, &st);
         exit(0)
      }

      ... any other application specific login valid actions ...
```

```
      }

   } else {
      sec_login_purge_context(&login_context, &st);

      ... application specific login failure actions ...
   }
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_set_context(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_valid_and_cert_ident

Purpose

Validates and certifies a login context

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_valid_and_cert_ident(
        sec_login_handle_t login_context,
        sec_passwd_rec_t *passwd,
        boolean32 *reset_passwd,
        sec_login_auth_src_t *auth_src,
        error_status_t *status);
```

Parameters

Input

login_context

An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

passwd A password record to be checked against the password in the principal's registry account. The routine returns TRUE if the two match. The contents of the *passwd* parameter are erased after the call has finished processing it.

Output

reset_passwd

A pointer to a 32-bit **boolean32** value. The routine returns TRUE if the account password has expired and must be reset.

auth_src A 32-bit set of flags identifying the source of the authentication. Upon return after successful authentication, the flags in *auth_src*

indicate what authority was used to validate the login context. If the authentication was accomplished with the network authority, the **sec_login_auth_src_network** flag is set, and the process login context has credentials to use the network.

If the authentication was accomplished with local data only (either the principal's account is tailored for the local machine with overrides, or the network authority is unavailable), the **sec_login_auth_src_local** flag is set. Login contexts that are authenticated locally may not be used to establish network credentials because they have none.

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_valid_and_cert_ident()** routine validates and certifies a login context established with **sec_login_setup_identity()**. The caller must supply the user's password as input with the *passwd* parameter.

This routine combines the operations of the **sec_login_validate_identity()** and **sec_login_certify_identity()** routines. It is intended for use by system login programs that need to extract trustworthy operating system credentials for use in setting the local identity for a process. This operation destroys the contents of the *passwd* input parameter.

If the network security service is unavailable or if the user's password has been overridden on the host, a locally authenticated context is created, and the *auth_src* parameter is set to **sec_login_auth_src_local**. Data extracted from a locally authenticated context may be used to set the local OS identity, but it cannot be used to establish network credentials.

This routine is a privileged operation.

Return Values

The routine returns TRUE if the login identity has been successfully validated.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_passwd_invalid
The input string does not match the account password.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

sec_login_s_acct_invalid
The account is invalid or has expired.

sec_login_s_privileged
This is a privileged operation and was invoked by an unprivileged process.

sec_login_s_null_password
The input string is NULL.

sec_login_s_default_use
The input context was the default context, which cannot be validated.

sec_login_s_already_valid
The login context has already been validated.

sec_login_s_unsupp_passwd_type
The password type is not supported.

sec_login_s_no_memory
Not enough memory is available to complete the operation.

sec_login_s_preauth_failed
Preauthentication failure.

sec_pk_e_domain_unsupported
The DCE login domain is not supported by the personal security mechanism.

sec_pk_e_device_error
Personal security mechanism device error.

sec_pk_e_usage_unsupported
A private key of the required type was not located in the personal security mechanism.

sec_pk_e_unauthorized
The password is invalid for personal security mechanism access.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_valid_and_cert_ident()** routine as part of a system login process:

```
if (sec_login_setup_identity(<user>,
     sec_login_no_flags, &login_context, &st)) {
         ... get password ...
         if (sec_login_valid_and_cert_ident(login_context,
              password, &st)) {
               if (auth_src == sec_login_auth_src_network) {
                  if (GOOD_STATUS(&st)
                  sec_login_set_context(login_context);
                  }
         }
         if (reset_passwd) {
            ... reset the user's password ...
            if (passwd_reset_fails) {
                sec_login_purge_context(login_context)
                ... application login failure actions ...
         }
         ... application specific login valid actions ...
     }
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_certify_identity(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_valid_from_keytable

Purpose

Validates a login context's identity using input from a specified keytable file

Synopsis

```
#include <dce/sec_login.h>

void sec_login_valid_from_keytable(
        sec_login_handle_t login_context,
        unsigned32 authn_service,
        void *arg,
        unsigned32 try_kvno,
        unsigned32 *used_kvno,
        boolean32 *reset_passwd,
        sec_login_auth_src_t *auth_src,
        error_status_t *status);
```

Parameters

Input

login_context

An opaque handle to login context data. The login context contains, among other data, the account principal's name and UUID, account restrictions, records of the account principal's group memberships, and the account's home directory. (See **sec_intro(3sec)** for more details about the login context.)

authn_service

Identifies the authentication protocol using the key. The possible authentication protocols are as follows:

rpc_c_authn_dce_secret
DCE shared-secret key authentication.

rpc_c_authn_dce_public
DCE public key authentication (reserved for future use).

arg

This parameter can specify either the local keytab file or an argument to the *get_key_fn* key acquisition routine of the **rpc_server_register_auth_info** routine.

A value of NULL specifies that the default keytab file should be used. A keytab filename specifies that that file should be used as the keytab file. You must prepend the file's absolute filename with **FILE:** and the file must have been created with the **rgy_edit** command or the **sec_key_mgmt_set_key** routine.

Any other value specifies an argument for the *get_key_fn* key acquisition routine. See the **rpc_server_register_auth_info()** reference page for more information.

try_kvno The version number of the key in the keytab file to try first. Specify NULL to try the current version of the key.

Output

used_kvno A pointer to a 32-bit **boolean32** value that specifies the version number of the the key from the keytab file that was used to successfully validate the login context, if any.

reset_passwd

A pointer to a 32-bit **boolean32** value. The routine returns TRUE if the account password has expired and should be reset.

auth_src How the the login context was authorized. The **sec_login_auth_src_t** data type distinguishes the various ways the login context was authorized. There are three possible values:

sec_login_auth_src_network

Authentication accomplished through the normal network authority. A login context authenticated this way will have all the network credentials it ought to have.

sec_login_auth_src_local

Authentication accomplished via local data. Authentication occurs locally if a principal's account is tailored for the local machine, or if the network authority is unavailable. Since a login contexts authenticated locally has no network credentials, it can not be used for network operations.

sec_login_auth_src_overridden
Authentication accomplished via the override facility.

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_valid_from_keytable ()** routine validates the login context established with **sec_login_setup_identity()**. The **sec_login_valid_from_keytable ()** routine obtains the principal's password from the specified keytable.

If *try_kvno* specifies a key version number, that version number key is tried first, otherwise the current key version number is tried first. The function trys all keys in the keytable until it finds one that validates the login context. This operation must be invoked before the network credentials can be used.

Notes

A context is not secure and must not be set or exported until the authentication service is itself authenticated with the **sec_login_certify_identity()** call.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_passwd_invalid
The input string does not match the account password.

sec_rgy_server_unavailable
There is no data with which to compare the input string.

sec_login_s_acct_invalid
The account is invalid or has expired.

sec_login_s_default_use
The input context was the default context, which cannot be validated.

sec_login_s_already_valid
The login context has already been validated.

sec_login_s_unsupp_passwd_type
The password type is not supported.

sec_key_mgmt_e_key_unavailable
The requested key is not present.

sec_key_mgmt_e_authn_invalid
The authentication protocol is not valid.

sec_key_mgmt_e_unauthorized
The caller is not authorized to perform the operation.

sec_s_no_memory
Unable to allocate memory.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_valid_from_keytable()** routine as part of a straightforward login process:

```
if (sec_login_setup_identity(user_name, sec_login_no_flags,
                          &login_context, &st)) {
   ... get password from local keytable...

   if (sec_login_valid_from_keytable(login_context, authn_service,
       arg, try_kvno, &used_kvno, &reset_passwd,
       &auth_src, &st)) {

      sec_login_set_context(login_context, &st);
```

```
      if (auth_src != sec_login_auth_src_network)
         printf("no network credentials");

      }

      ... any other application specific login valid actions ...
      }

   } else {
      sec_login_purge_context(&login_context, &st);

      ... application specific login failure actions ...
   }
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_certify_identity(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_valid_and_cert_ident(3sec)**, **sec_login_validate_identity(3sec)**.

sec_login_validate_first

Purpose

Validates the initial login context

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_validate_first(
        sec_login_handle_t init_context,
        boolean32 *reset_passwd,
        sec_login_auth_src_t *auth_src,
        error_status_t *status);
```

Parameters

Input

init_context

An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. In this call, the context will be that of the host machine initial process. (See **sec_intro(3sec)** for more details about the login context.)

Output

reset_passwd

A pointer to a 32-bit **boolean32** value. The routine returns TRUE if the account password has expired and must be reset.

auth_src

A 32-bit set of flags identifying the source of the authentication. Upon return after successful authentication, the flags in *auth_src* indicate what authority was used to validate the login context. If the authentication was accomplished with the network authority, the **sec_login_auth_src_network** flag is set, and the process login context has credentials to use the network. If the authentication was accomplished with local data only (either the principal's account is

tailored for the local machine with overrides, or the network authority is unavailable), the **sec_login_auth_src_local** flag is set. Login contexts that are authenticated locally may not be used to establish network credentials because they have none.

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_validate_first()** routine validates the default login context established via **sec_login_setup_first()**. Typically, this operation is called from the security validation service of the **dced** process to validate the default credentials for the host machine process hierarchy. This operation uses the password for the local host, and therefore does not require a password parameter.

Return Values

The routine returns a **boolean32** value that is TRUE if the setup was successful, and FALSE otherwise.

Files

/usr/include/dce/sec_login.idl
The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_privileged
An unprivileged process was called in.

sec_rgy_server_unavailable
The network authentication service was unavailable.

sec_pk_e_domain_unsupported
The DCE login domain is not supported by the personal security mechanism.

sec_pk_e_device_error
Personal security mechanism device error.

sec_pk_e_usage_unsupported
A private key of the required type was not located in the personal security mechanism.

sec_pk_e_unauthorized
The password is invalid for personal security mechanism access.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_login_init_first(3sec)**, **sec_login_setup_first(3sec)**.

sec_login_validate_identity

Purpose

Validates a login context's identity

Synopsis

```
#include <dce/sec_login.h>

boolean32 sec_login_validate_identity(
        sec_login_handle_t login_context,
        sec_passwd_rec_t *passwd,
        boolean32 *reset_passwd,
        sec_login_auth_src_t *auth_src,
        error_status_t *status);
```

Parameters

Input

login_context
: An opaque handle to login context data. The login context contains, among other data, the account principal name and UUID, account restrictions, records of group membership, and the process home directory. (See **sec_intro(3sec)** for more details about the login context.)

passwd
: A password record to be checked against the password in the principal's registry account. The routine returns TRUE if the two match. The contents of the *passwd* parameter are erased after the call has finished processing it.

Output

reset_passwd
: A pointer to a 32-bit **boolean32** value. The routine returns TRUE if the account password has expired and must be reset.

auth_src How the the login context was authorized. The **sec_login_auth_src_t** data type distinguishes the various ways the login context was authorized. There are three possible values:

- **sec_login_auth_src_network**
- **sec_login_auth_src_local**
- **sec_login_auth_src_overridden**

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_login_validate_identity()** routine validates the login context established with **sec_login_setup_identity()**. This operation must be invoked before the network credentials can be used. The caller must supply the principal's password in a **sec_passwd_rec_t** as input with the *passwd* parameter. The following example sets up a plaintext password for the the *passwd* parameter:

```
sec_passwd_str_t      tmp_passwd;

passwd.version_number = sec_passwd_c_version_none;
passwd.pepper = NULL;
passwd.key.key_type = sec_passwd_plain;

strncpy((char *) tmp_passwd, (char *) my_passwd,
        sec_passwd_str_max_len);
tmp_passwd[sec_passwd_str_max_len] = ' ';
passwd_rec.key.tagged_union.plain = &(tmp_passwd[0]);
```

When a network identity is set, only state information for network operations has been established. The local operating system identity has not been modified. It is the responsibility of the caller to establish any local operating identity state.

The **sec_login_setup_identity()** operation and the **sec_login_validate_identity()** operation are two halves of a single logical operation. Together they collect the

identity data needed to establish an authenticated identity. The operations are independent so the principal's password need not be sent across the network. The identity validation performed by **sec_login_validate_identity()** is a local operation.

Notes

A context is not secure and must not be set or exported until the authentication service is itself authenticated with the **sec_login_certify_identity()** call.

System login programs that set local operating system identity using data extracted from a login context should use **sec_login_valid_and_cert_ident()** instead of **sec_login_validate_identity()**.

If the security server and client clocks are not synchronized to within 2 to 3 minutes of each other, this call can return a password validation error.

Return Values

The routine returns TRUE if the login identity has been successfully validated.

Files

/usr/include/dce/sec_login.idl
: The **idl** file from which **dce/sec_login.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_passwd_invalid
: The input string does not match the account password.

sec_rgy_server_unavailable
: There is no data with which to compare the input string.

sec_login_s_acct_invalid
: The account is invalid or has expired.

sec_login_s_null_password
The input string is NULL.

sec_login_s_default_use
The input context was the default context, which cannot be validated.

sec_login_s_already_valid
The login context has already been validated.

sec_login_s_unsupp_passwd_type
The password type is not supported.

sec_login_s_no_memory
Not enough memory is available to complete the operation.

sec_login_s_preauth_failed
Preauthentication failure.

sec_pk_e_domain_unsupported
The DCE login domain is not supported by the personal security mechanism.

sec_pk_e_device_error
Personal security mechanism device error.

sec_pk_e_usage_unsupported
A private key of the required type was not located in the personal security mechanism.

sec_pk_e_unauthorized
The password is invalid for personal security mechanism access.

error_status_ok
The call was successful.

Examples

The following example illustrates use of the **sec_login_validate_identity()** routine as part of a straightforward login process:

```
if (sec_login_setup_identity(user_name, sec_login_no_flags,
                          &login_context, &st)) {
   ... get password from user...
```

```
   if (sec_login_validate_identity(login_context, password,
                           &reset_passwd, &auth_src, &st)) {

      if (!sec_login_certify_identity(login_context, &st))
          exit(error_weird_auth_svc);

      sec_login_set_context(login_context, &st);

      if (auth_src != sec_login_auth_src_network)
         printf("no network credentials");

      if (reset_passwd) {
         ... get new password from user, reset registry record ...
      };

      sec_login_get_pwent(login_context, &pw_entry, &st);

      if (pw_entry.pw_expire < todays_date) {
         sec_login_purge_context(&login_context, &st);
         exit(0)
      }

      ... any other application specific login valid actions ...
      }

   } else {
      sec_login_purge_context(&login_context, &st);

      ... application specific login failure actions ...
   }
}
```

Related Information

Functions: **sec_intro(3sec)**, **sec_login_certify_identity(3sec)**, **sec_login_setup_identity(3sec)**, **sec_login_valid_and_cert_ident(3sec)**.

sec_pk_data_free

Purpose

Frees memory allocated to a **sec_pk_data_t** and its aliases. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <sec_pk_base.h>

void sec_pk_data_free(
        sec_pk_data_t *data_p);
```

Parameters

Input/Output

data_p As input, a pointer to a **sec_pk_data_t** that points to the memory to be reclaimed. As output, a pointer to a **sec_pk_data_t** that is set to NULL.

Description

The **sec_pk_data_free()** routine frees and sets to NULL each nonnull pointer in a **sec_pk_data_t**. Use this function, rather than **sec_pk_data_zero_and_free()**, for **sec_pk_data_t** structures that contain a public key pair and other nonsensitive data.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Related Information

Functions: **sec_pk_data_zero_and_free(3sec)**.

sec_pk_data_zero_and_free

Purpose

Frees and zeros out memory allocated to a **sec_pk_data_t** or its aliases. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

#include <sec_pk_base.h>

void sec_pk_data_zero_and_free(
sec_pk_data_t ****data_p*);**

Parameters

Input/Output

data_p As input, a pointer to a **sec_pk_data_t** that points to the memory to be reclaimed. As output, a pointer to a **sec_pk_data_t** that is set to NULL.

Description

The **sec_pk_data_zero_and_free()** routine zeros out and frees memory allocated to a **sec_pk_data_t** or its aliases. Use this function, rather than **sec_pk_data_free()**, for structures that contain the private part of a public key pair or secret keys.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Related Information

Functions: **sec_pk_data_free(3sec)**.

sec_psm_close

Purpose

Close a personal security mechanism. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_close(
        sec_psm_handle_t psm_handle,
        error_status_t *status);
```

Parameters

Input

psm_handle A pointer to an opaque handle to the personal security context data. Use the **sec_psm_open()** routine to obtain the handle.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_close()** routine closes the personal security mechanism identified by *psm_handle*. In addition, the routine cleans up the personal security context data and ensures any confidential information (such as passwords or private keys) is zeroed out.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_t

sec_psm_not_init

sec_psm_invalid_handle

sec_psm_internal_error

sec_pvtkey_invalid_handle

sec_pvtkey_mechanism_not_init

Related Information

Functions: **sec_psm_open(3sec)**.

sec_psm_decrypt_data

Purpose

Decrypt data that was encrypted with a public key mechanism. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_decrypt_data(
        sec_psm_handle_t psm_handle,
        unsigned32 *kvno,
        sec_pk_algorithm_id_t *encryption_alg_id,
        sec_pk_usage_flags_t key_usage,
        sec_pk_encrypted_t *cipher_data,
        sec_pk__gen_data_t *clear_data,
        error_status_t *status);
```

Parameters

Input

psm_handle
: A pointer to an opaque handle to the personal security mechanism context. Use **sec_psm_open()** to obtain the handle.

kvno
: The unsigned 32-bit version number of the key in which the data is encrypted.

encryption_alg_id
: The ASN.1 DER-encoded object ID of the encryption algorithm, such as RSA, used to encrypt the data.

key_usage
: A **sec_pk_usage_flags_t** that contains the usage flag for the private key to be used for this operation.

cipher_data
A pointer to a **sec_pk_data_t** that contains the ASN.1 DER-encoded data to be decrypted.

Output

clear_data
A pointer to the decrypted data.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_decrypt_data()** routine decrypts data that was encrypted with a public key mechanism. The *encryption_alg_id* parameter specifies the encryption algorithm used. This routine allocates memory for the returned decrypted data. Call the **sec_pk_data_free()** routine to deallocate that memory.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_psm_unsupported_algorithm_id

sec_bsafe_encryption_failure

sec_pvtkey_invalid_handle

sec_pvtkey_mechanism_not_init

sec_pvtkey_internal_error

sec_pvtkey_invalid_password

sec_pvtkey_multiple_key_usage

Related Information

Functions: **sec_pk_data_free(3sec)**, **sec_psm_encrypt_data(3sec)**.

sec_psm_encrypt_data

Purpose

Encrypt data using a public key mechanism. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_encrypt_data(
        sec_psm_handle_t psm_handle,
        sec_pk domain_t *encryptee_domain,
        void *encryptee_name,
        unsigned32 *kvno,
        sec_pk_algorithm_id_t *encryption_alg_id,
        sec_pk_usage_flags_t key_usage,
        sec_pk_gen_data_t *clear_data,
        sec_pk_encrypted_t **cipher_data,
        error_status_t *status);
```

Parameters

Input

psm_handle
: A pointer to an opaque handle to the personal security context data. Use **sec_psm_open()** to obtain the handle.

encryptee_domain
: A pointer to the application domain of the principal for which the data is encrypted.

encryptee_name
: A pointer to the name of the principal for which the data is encrypted.

encryption_alg_id
: The ASN.1 DER-encoded object ID of the encryption algorithm to use, such as RSA.

key_usage A **sec_pk_usage_flags_t** that contains the usage flag for the public key specified by *data*.

clear_data A pointer to ASN.1 DER-encoded data to be encrypted.

Input/Output

kvno As input, the unsigned 32-bit version number of the key with which to encrypt the data. As output, the unsigned 32-bit version number of the key used to encrypt the data.

Output

cipher_data
A pointer to the encrypted data.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_encrypt_data()** routine encrypts data by using a public key encryption mechanism. The *encryption_alg_id* parameter specifies the encryption algorithm. This routine allocates memory for *cipher_data*. Call the **sec_pk_data_free()** routine to deallocate that memory.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_psm_unsupported_algorithm_id

sec_pk_e_domain_unsupported

sec_pk_e_usage_unsupported

sec_rgy_object_not_found

sec_rgy_not_authorized

sec_attr_unsupported

Related Information

Functions: **sec_pk_data_free(3sec)**, **sec_psm_decrypt_data(3sec)**.

sec_psm_gen_pub_key

Purpose

Randomly generate a public key pair. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_gen_pub_key(
        sec_pk_algorithm_id_t *key_type,
        unsigned32 modulus_bit_size,
        sec_pk_gen_data_t *seed,
        sec_pk_data_t *public_key,
        sec_pk_data_t *private_key,
        error_status_t *status);
```

Parameters

Input

key_type A pointer to the object ID of the public key encryption algorithm to use. Only the RSA public key algorithm (RSA_PKCS) is currently supported.

modulus_bit_size
The desired length of the key. Interpretation of this parameter is dependent on the algorithm specified by *key_type*. For RSA, the only currently supported key type, *modulus_bit_size* is a number ranging from 256 through 1024 inclusive that specifies the bit length of the key modulus. A value of **0** indicates the default of 1024.

seed A pointer to the string to seed the random key generator.

Output

private_key
A pointer to a **sec_pk_data_t** that contains the private key structure of the newly generated key.

public_key
A pointer to a **sec_pk_data_t** that contains the public key structure of the newly generated key.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_gen_pub_key()** routine generates a public key pair. This routine allocates memory for the returned key. Call the **sec_pk_data_free()** routine to deallocate the public key and **sec_pk_data_zero_and_free()** routine to deallocate the private key.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_wrong_pub_key_type

sec_bsafe_alloc

Related Information

Functions: **sec_psm_update_pub_key(3sec)**, **sec_psm_put_pub_key(3sec)**.

sec_psm_open

Purpose

Open a personal security mechanism. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_open(
        void *name,
        char *pwd,
        sec_pk_domain_t *domain_id,
        sec_psm_handle_t *psm_handle,
        error_status_t *status);
```

Parameters

Input

name A pointer to the name of the principal for which to open the personal security mechanism. Supply this name in the form **/.:/principal_name** or **/.../cell_name/principal_name**/.

pwd A pointer to the principal's password.

domain_id A pointer to the application domain that the principal is operating on. (Currently, the only domain supported is **sec_pk_domain_dce_pk_login**.)

Output

psm_handle A pointer to an opaque handle to the personal security context data.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_open()** routine obtains a handle to a personal security mechanism for the principal specified by *name* by using the password specified with *pwd***.**

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_pvtkey_privileged

sec_pvtkey_no_more_memory

sec_psm_no_more_memory

Related Information

Functions: **sec_psm_close(3sec)**.

sec_psm_put_pub_key

Purpose

Store a public key pair . This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_put_pub_key(
        sec_psm_handle_t psm_handle,
        char *pwd,
        sec_pk_usage_flags_t key_usage,
        sec_pk_pvtkey_t *pvtkey,
        sec_pk_pvtkey_t *pubkey,
        error_status_t *status);
```

Parameters

Input

psm_handle
: A pointer to an opaque handle to the personal security mechanism in which to store the key. Use **sec_psm_open()** to obtain the handle.

pwd
: A pointer to the password for the principal associated with the personal security mechanism.

key_usage
: A **sec_pk_usage_flags_t** that contains the usage flag for the public key for the key pair specified by *pubkey*.

pvtkey
: A pointer to the ASN.1 DER-encoded private key.

pubkey
: A pointer to the ASN.1 DER-encoded public key.

Output

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_put_pub_key()** routine stores a public key pair. In the reference implementation, the public key is stored in the registry and the private key in a personal security mechanism. Key versions are not currently supported; only a single version of a key with a given key usage is maintained.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_pk_e_domain_unsupported

sec_pk_e_usage_unsupported

sec_rgy_object_not_found

sec_rgy_not_authorized

sec_attr_unauthorized

sec_pvtkey_invalid_handle

sec_pvtkey_mechanism_not_init

sec_pvtkey_no_more_memory

sec_pvtkey_internal_error

sec_pvtkey_same_domain_and_usage_key_already_exists

Related Information

Functions: **sec_psm_gen_pub_key(3sec)**, **sec_psm_update_pub_key(3sec)**.

sec_psm_sign_data

Purpose

Compute the signature of data using a specified signature algorithm . This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_sign_data(
        sec_psm_handle_t psm_handle,
        sec_pk_algorithm_id_t *signature_alg_id,
        sec_pk_usage_flags_t key_usage,
        sec_pk_gen_data_t *data,
        unsigned32 *kvno,
        sec_pk_signed_t *signature,
        error_status_t *status_t);
```

Parameters

Input

psm_handle
A pointer to an opaque handle to the personal security context data. Use **sec_psm_open()** to obtain the handle.

signature_alg_id
The ASN.1 DER-encoded object ID of the signature algorithm. **MD5WithRSAEncryption** is the only algorithm ID currently supported.

key_usage
A **sec_pk_usage_flags_t** that contains the usage flag of the private key to be used in this operation.

data A pointer to the ASN.1 DER-encoded data to be signed.

Output

kvno The version of the key being used.

signature A pointer to the computed signature.

status_t A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_sign_data()** routine computes the signature of input data by using the signature algorithm specified by *signature_alg_id*. This routine allocates memory for the returned signed data. Call the **sec_pk_data_free()** routine to deallocate that memory.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_psm_unsupported_algorithm_id

sec_pvtkey_invalid_handle

sec_pvtkey_mechanism_not_init

sec_pvtkey_internal_error

sec_pvtkey_invalid_password

sec_pvtkey_multiple_key_usages

Related Information

Functions: **sec_pk_data_free(3sec)**, **sec_psm_verify_data(3sec)**.

sec_psm_update_pub_key

Purpose

Update a public key in a personal security mechanism. . This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_update_pub_key(
        sec_psm_handle_t psm_handle,
        char *oldpwd,
        char *newpwd,
        sec_pk_usage_flags_t key_usage,
        sec_pk_pubkey_t *pubkey,
        sec_pk_pvtkey_t *pvtkey,
        error_status_t *status);
```

Parameters

Input

psm_handle
: A pointer to an opaque handle to a personal security mechanism context data. Use **sec_psm_open()** to obtain the handle.

oldpwd
: A pointer to the principal's current password.

newpwd
: A pointer to the principal's new password.

key_usage
: A **sec_pk_usage_flags_t** that contains the usage flag for the public key in the key pair.

pubkey
: A pointer to the ASN.1 DER-encoded public key.

pvtkey
: A pointer to the ASN.1 DER-encoded private key.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_update_pub_key()** routine updates a principal's public key pair or password. The current public key password must be supplied for authentication. Currently, only a single version of a key with a given key usage is maintained. Therefore, any old key versions are overwritten by this routine. Note that there is no routine supplied to delete keys; deletion is assumed to be an internal function initiated by the personal security mechanism.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_pk_e_domain_unsupported

sec_pk_e_usage_unsupported

sec_rgy_object_not_found

sec_rgy_not_authorized

sec_attr_unauthorized

sec_pvtkey_invalid_handle

sec_pvtkey_mechanism_not_init

sec_pvtkey_private_key_is_not_supplied

sec_pvtkey_new_password_required

sec_pvtkey_no_more_memory

Related Information

Functions: **sec_psm_gen_pub_key(3sec)**, **sec_psm_put_pub_key(3sec)**.

sec_psm_verify_data

Purpose

Verify signed data. This routine is not available in the DCE binary code. It is provided in DCE source for use by vendors.

Synopsis

```
#include <dce/sec_pk_base.h>

error_status_t sec_psm_verify_data(
        sec_psm_handle_t psm_handle,
        sec_pk_domain_t *signer_domain_id,
        void *signer_name,
        unsigned32 *kvno,
        sec_pk_algorithm_id_t *signature_alg_id,
        sec_pk_usage_flags_t key_usage,
        sec_pk_gen_data_t *data,
        sec_pk_signed_t *signature,
        error_status_t *status);
```

Parameters

Input

psm_handle
: A pointer to an opaque handle to personal security context data. Use **sec_psm_open()** to obtain the handle.

signer_domain_id
: A pointer to the application domain of the principal that signed the data.

signer_name
: A pointer to the name of the principal that signed the data.

kvno
: The version of the key being used.

signature_alg_id
The ASN.1 DER-encoded object ID of the signature algorithm, such as **MD5WithRSAEncryption**.

key_usage A **sec_pk_usage_flags_t** that contains the usage flag for the public key.

data A pointer to the data to be verified.

signature A pointer to the signature to be verified.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_psm_sign_data()** routine verifies input data, usually the data signature of input data.

Files

/usr/include/dce/sec_pk_base.idl
The **idl** file from which **dce/sec_pk_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

sec_psm_not_init

sec_psm_invalid_handle

sec_psm_unsupported_algorithm_id

sec_pk_e_domain_unsupported

sec_rgy_object_not_found

sec_rgy_not_authorized

sec_attr_unauthorized

Related Information

Functions: **sec_psm_sign_data(3sec)**.

sec_pwd_mgmt_free_handle

Purpose

Frees storage allocated for a password management handle

Synopsis

```
#include <dce/sec_pwd_mgmt.h>

void sec_pwd_mgmt_free_handle(
        sec_pwd_mgmt_handle_t *pwd_mgmt_h,
        error_status_t *stp);
```

Parameters

Input/Output

pwd_mgmt_h
: A handle to the password management data which is to be freed.

Output

stp
: A pointer to the completion status. On successful completion, the routine returns error_status_ok. Otherwise, it returns an error.

Description

The **sec_pwd_mgmt_free_handle()** routine frees any memory allocated for the contents of a password management handle.

Files

/usr/include/dce/sec_pwd_mgmt.idl
: The idl file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful

Related Information

Functions: **sec_intro(3sec)**, **sec_pwd_mgmt_setup(3sec)**.

sec_pwd_mgmt_gen_pwd

Purpose

Generates a set of passwords

Synopsis

```
#include <dce/sec_pwd_mgmt.h>

void sec_pwd_mgmt_gen_pwd(
        sec_pwd_mgmt_handle_t pwd_mgmt_h,
        unsigned32 num_pwds,
        unsigned32 *num_returned,
        sec_passwd_rec_t gen_pwds[ ],
        error_status_t *stp);
```

Parameters

Input

pwd_mgmt_h
: A handle to user's password management data.

num_pwds
: Number of generated passwords requested.

Output

num_returned
: Number of generated passwords returned in the *gen_pwds*[ ] array.

gen_pwds[ ]
: Array of generated passwords. Each generated password is stored in a **sec_passwd_rec_t** structure.

stp
: A pointer to the completion status. On successful completion, status is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_pwd_mgmt_gen_pwd()** routine retrieves a set of generated passwords from a password management server which is exporting the **rsec_pwd_mgmt_gen_pwd()** routine. It obtains the binding information to this server from the *pwd_mgmt_h* handle.

Files

/usr/include/dce/sec_pwd_mgmt.idl
The idl file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_era_pwd_mgmt_auth_type
The pwd_mgmt_binding ERA must contain authentication information.

sec_pwd_mgmt_svr_unavail
The password management server is unavailable.

sec_pwd_mgmt_svr_error
Generic error returned from password management server. An administrator should check the password management server's log file for more information.

error_status_ok
The call was successful

Various RPC communication errors can be returned if there are failures when binding to the password management server.

Related Information

Functions: **pwd_strengthd(8sec)**, **sec_intro(3sec)**, **sec_pwd_mgmt_setup(3sec)**.

sec_pwd_mgmt_get_val_type

Purpose

Gets users password validation type

Synopsis

```
#include <dce/sec_pwd_mgmt.h>

void sec_pwd_mgmt_get_val_type(
        sec_pwd_mgmt_handle_t pwd_mgmt_h,
        signed32 *pwd_val_type,
        error_status_t *stp);
```

Parameters

Input

pwd_mgmt_h
: A handle to a user's password management data.

Output

pwd_val_type
: The user's password validation type. This is retrieved from the *pwd_val_type* ERA. The possible values and their meaning are as follows:

  0 (**none**): the user has no password policy.

  1 (**user_select**): the user must choose his/her own password.

  2 (**user_can_select**): the user can choose his/her own password or request a generated password.

  3 (**generation_required**): the user must use a generated password.

stp
: A pointer to the completion status. On successful completion, stp is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_pwd_mgmt_get_val_type()** routine returns the value of the user's password validation type, as specified by the *pwd_val_type* ERA. If the ERA does not exist, **0** (**none**) is returned in *pwd_val_type*.

Files

/usr/include/dce/sec_pwd_mgmt.idl
The idl file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Various RPC communication errors can be returned if there are failures when binding to the password management server.

Related Information

Functions: **sec_intro(3sec)**, **sec_pwd_mgmt_setup(3sec)**.

sec_pwd_mgmt_setup

Purpose

Sets up the user's password policy information

Synopsis

```
#include <dce/sec_pwd_mgmt.h>

void sec_pwd_mgmt_setup(
        sec_pwd_mgmt_handle_t *pwd_mgmt_h,
        sec_rgy_handle_t context,
        sec_rgy_login_name_t login_name,
        sec_login_handle_t your_lc,
        rpc_binding_handle_t pwd_mgmt_bind_h,
        error_status_t *stp);
```

Parameters

Input

context A registry server handle indicating the desired registry site.

login_name The login name of the user.

your_lc The login context handle of the user currently logged in. If null is specified, the default login context will be used.

pwd_mgmt_bind_h
An RPC binding handle to the password management server. Use of this parameter is currently unsupported. The password management server binding handle will be retrieved from the **pwd_mgmt_binding** ERA. Set this parameter to NULL.

Output

pwd_mgmt_h
A pointer to an opaque handle to password management/policy data. *pwd_mgmt_h* contains, among other data, the account name, values of

password management ERAs, and a binding handle to the password management server.

stp A pointer to the completion status. On successful completion, *stp* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_pwd_mgmt_setup()** routine collects the data required to perform remote password management calls to the password management server.

Files

/usr/include/dce/sec_pwd_mgmt.idl
The idl file from which **dce/sec_pwd_mgmt.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_s_no_memory
Not enough memory is available to complete the operation.

sec_rgy_server_unavailable
The network registry is not available.

error_status_ok
The call was successful.

Related Information

Functions: **pwd_strengthd(8sec)**, **sec_intro(3sec)**, **sec_pwd_mgmt_free_handle(3sec)**, **sec_pwd_mgmt_gen_pwd(3sec)**, **sec_pwd_mgmt_get_val_type(3sec)**.

sec_rgy_acct_add

Purpose

Adds an account for a login name

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_add(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_acct_user_t *user_part,
        sec_rgy_acct_admin_t *admin_part,
        sec_passwd_rec_t *caller_key,
        sec_passwd_rec_t *new_key,
        sec_passwd_type_t new_keytype,
        sec_passwd_version_t *new_key_version,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. All three names must be completely specified.

key_parts
A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

user_part A pointer to the **sec_rgy_acct_user_t** structure containing the user part of the account data. This represents such information as the account password, home directory, and default shell.

admin_part A pointer to the **sec_rgy_acct_admin_t** structure containing the administrative part of an account's data. This information includes the account creation and expiration dates and flags describing limits to the use of privilege attribute certificates, among other information.

caller_key
The key representing the user's current password, used to encrypt *new_key* for transmission to the registry server.

new_key The password for the new account. During transmission to the registry server, it is encrypted with *caller_key*.

new_keytype
The type of the new key. The server uses this parameter to decide how to encode *new_key* if it is sent as plaintext.

Output

new_key_version
The key version number returned by the server. If the client requests a particular key version number (via the *version_number* field of the *new_key* input parameter), the server returns the requested version number back to the client.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_add()** routine adds an account with the specified login name. The login name is given in three parts, corresponding to the principal, group, and organization names for the account. All input paramters and all fields in those parameters are required.

The *key_parts* variable specifies the minimum login abbreviation for the account. If the requested abbreviation duplicates an existing abbreviation for another account, the routine supplies the next shortest unique abbreviation and returns this abbreviation in *key_parts*. Abbreviations are not currently implemented.

Permissions Required

The **sec_rgy_acct_add()** routine requires the following permissions on the account (principal) that is to be added:

- The **m** (**mgmt_info**) permission to change management information.
- The **a** (*auth_info*) permission to change authentication information.
- The **u** (**user_info**) permission to change user information.

Notes

The constituent principal, group, and organization (PGO) items for an account must be added before the account can be created. (See the **sec_rgy_pgo_add()** routine). Also, the principal must have been added as a member of the specified group and organization. (See the **sec_rgy_pgo_add_member()** routine).

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to add an account to the registry.

sec_rgy_not_member_group
The indicated principal is not a member of the indicated group.

sec_rgy_not_member_org
The indicated principal is not a member of the indicated organization.

sec_rgy_not_member_group_org
The indicated principal is not a member of the indicated group or organization.

sec_rgy_object exists
The account to be added already exists.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_delete(3sec)**, **sec_rgy_login_get_info(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_add_member(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_acct_admin_replace

Purpose

Replaces administrative account data

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_admin_replace(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_acct_admin_t *admin_part,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name

A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. For the group and organization names, blank strings can serve as wildcards, matching any entry. The principal name must be input.

key_parts

A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

admin_part

A pointer to the **sec_rgy_acct_admin_t** structure containing the administrative part of an account's data. This information includes the account creation and expiration dates and flags describing limits to

the use of privilege attribute certificates, among other information, and can be modified only by an administrator. The **sec_rgy_acct_admin_t** structure contains the following fields:

creator The identity of the principal who created this account in **sec_rgy_foreign_id_t** form. This field is set by the registry server.

creation_date
The date (**sec_timeval_sec_t**) the account was created. This field is set by the registry server.

last_changer
The identity of the principal who last modified any of the account information (user or administrative). This field is set by the registry server.

change_date
The date (**sec_timeval_sec_t**) the account was last modified (either user or administrative data). This field is set by the registry server.

expiration_date
The date (**sec_timeval_sec_t**) the account will cease to be valid.

good_since_date
This date (**sec_timeval_sec_t**) is for Kerberos-style, ticket-granting ticket revocation. Ticket-granting tickets issued before this date will not be honored by authenticated network services.

flags Contains administration flags used as part of the administrator's information for any registry account. This field is in **sec_rgy_acct_admin_flags_t** form. (See **sec_intro(3sec)** for a complete description of these flags.)

authentication_flags
Contains flags controlling use of authentication services. This field is in **sec_rgy_acct_auth_flags_t** form. (See **sec_intro(3sec)** for a complete description of these flags.)

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_admin_replace()** routine replaces the administrative information in the account record specified by the input login name. The administrative information contains limitations on the account's use and privileges. It can be modified only by a registry administrator; that is, a user with the **admin_info** (abbreviated as **a**) privilege for an account.

The *key_parts* variable identifies how many of the *login_name* parts to use as the unique abbreviation for the account. If the requested abbreviation duplicates an existing abbreviation for another account, the routine supplies the next shortest unique abbreviation and returns this abbreviation using *key_parts*.

Permissions Required

The **sec_rgy_acct_admin_replace()** routine requires the following permissions on the account principal:

- The **m** (**mgmt_info**) permission, if **flags** or **expiration_date** is to be changed.
- The **a** (*auth_info*) permission, if **authentication_flags** or **good_since_date** is to be changed.

Notes

All users need the **w** (**write**) privilege in the appropriate ACL entry to modify any account information.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to change the administrative information for the specified account.

sec_rgy_object_not_found
The registry server could not find the specified name.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_lookup(3sec)**, **sec_rgy_acct_replace_all(3sec)**, **sec_rgy_acct_user_replace(3sec)**.

sec_rgy_acct_delete

Purpose

Deletes an account

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_delete(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. Only the principal name is required to perform the deletion.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_delete()** routine deletes from the registry the account corresponding to the specified login name.

Permissions Required

The **sec_rgy_acct_delete()** routine requires the following permissions on the account principal:

- The **m** (**mgmt_info**) permission to remove management information.
- The **a** (*auth_info*) permission to remove authentication information.
- The **u** (**user_info**) permission to remove user information.

Notes

Even though the account is deleted, the PGO items corresponding to the account remain. These must be deleted with separate calls to **sec_rgy_pgo_delete()**.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to delete the specified account.

sec_rgy_object_not_found
No PGO item was found with the given name.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_pgo_delete(3sec)**.

sec_rgy_acct_get_projlist

Purpose

Returns the projects in an account's project list

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_get_projlist(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_cursor_t *projlist_cursor,
        signed32 max_number,
        signed32 *supplied_number,
        uuid_t id_projlist[ ],
        signed32 unix_projlist[ ],
        signed32 *num_projects,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name
A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. For the group and organization names, blank strings can serve as wildcards, matching any entry. The principal name must be input.

max_number
The maximum number of projects to be returned by the call. This must be no larger than the allocated size of the *projlist[ ]* arrays.

Input/Output

projlist_cursor
: An opaque pointer indicating a specific project in an account's project list. The **sec_rgy_acct_get_projlist()** routine returns the project indicated by *projlist_cursor*, and advances the cursor to point to the next project in the list. When the end of the list is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

supplied_number
: A pointer to the actual number of projects returned. This will always be less than or equal to the *max_number* supplied on input. If there are more projects in the account list, **sec_rgy_acct_get_projlist()** sets *projlist_cursor* to point to the next entry after the last one in the returned list.

id_projlist[ ]
: An array to receive the UUID of each project returned. The size allocated for the array is given by *max_number*. If this value is less than the total number of projects in the account project list, multiple calls must be made to return all of the projects.

unix_projlist[ ]
: An array to receive the UNIX number of each project returned. The size allocated for the array is given by *max_number*. If this value is less than the total number of projects in the account project list, multiple calls must be made to return all of the projects.

num_projects
: A pointer indicating the total number of projects in the specified account's project list.

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_get_projlist()** routine returns members of the project list for the specified account. It returns the project information in two arrays. The *id_projlist[ ]*

array contains the UUIDs for the returned projects. The *unix_projlist[ ]* array contains the UNIX numbers for the returned projects.

The project list cursor, *projlist_cursor*, provides an automatic place holder in the project list. The **sec_rgy_acct_get_projlist()** routine automatically updates this variable to point to the next project in the project list. To return an entire project list, reset *projlist_cursor* with **sec_rgy_cursor_reset()** on the initial call and then issue successive calls until all the projects are returned.

Permissions Required

The **sec_rgy_acct_get_projlist()** routine requires the **r** (**read**) permission on the account principal for which the project list data is to be returned.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to refresh a cursor for use with another call or for another server.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of projects.

sec_rgy_not_authorized
The client program is not authorized to see a project list for this principal.

sec_rgy_object exists
The account to be added already exists.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_get_next(3sec)**.

sec_rgy_acct_lookup

Purpose

Returns data for a specified account

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_lookup(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *name_key,
        sec_rgy_cursor_t *account_cursor,
        sec_rgy_login_name_t *name_result,
        sec_rgy_sid_t *id_sid,
        sec_rgy_unix_sid_t *unix_sid,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_acct_user_t *user_part,
        sec_rgy_acct_admin_t *admin_part,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_key A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. Blank strings serve as wildcards, matching any entry.

Input/Output

account_cursor
An opaque pointer to a specific account in the registry database. If *name_key* is blank, **sec_rgy_acct_lookup()** returns information about the account to which the cursor is pointing. On return, the cursor

points to the next account in the database after the returned account. If *name_key* is blank and the *account_cursor* has been reset with **sec_rgy_cursor_reset()**, **sec_rgy_acct_lookup()** returns information about the first account in the database.

When the end of the list of accounts in the database is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to refresh the cursor.

Output

name_result
A pointer to the full login name of the account (including all three names) for which the information is returned. The remaining parameters contain the information belonging to the returned account.

id_sid A structure containing the three UUIDs of the principal, group, and organization for the account.

unix_sid A structure containing the three UNIX numbers of the principal, group, and organization for the account.

key_parts A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

user_part A pointer to the **sec_rgy_acct_user_t** structure containing the user part of the account data. This represents such information as the account password, home directory, and default shell, all of which are accessible to, and may be modified by, the account owner.

admin_part A pointer to the **sec_rgy_acct_admin_t** structure containing the administrative part of an account's data. This information includes the account creation and expiration dates and flags describing limits to the use of privilege attribute certificates, among other information, and can be modified only by an administrator.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_lookup()** routine returns all the information about an account in the registry database. The account can be specified either with *name_key* or

account_cursor. If *name_key* is completely blank, the routine uses the *account_cursor* value instead.

For *name_key*, a zero-length principal, group, or organization key serves as a wildcard. For example, a login name key with the principal and organization fields blank returns the next (possibly first) account whose group matches the input group field. The full login name of the returned account is passed back in *name_result*.

The *account_cursor* provides an automatic place holder in the registry database. The routine automatically updates this variable to point to the next account in the database, after the account for which the information was returned. If *name_key* is blank and the *account_cursor* has been reset with **sec_rgy_cursor_reset()**, **sec_rgy_acct_lookup()** returns information about the first account in the database.

Permissions Required

The **sec_rgy_acct_lookup()** routine requires the **r** (**read**) permission on the account principal to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the accounts in the registry.

sec_rgy_object_not_found
The input account could not be found by the registry server.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_admin_replace(3sec)**, **sec_rgy_acct_replace_all(3sec)**, **sec_rgy_acct_user_replace(3sec)**, **sec_rgy_cursor_reset(3sec)**.

sec_rgy_acct_passwd

Purpose

Changes the password for an account

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_passwd(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_passwd_rec_t *caller_key,
        sec_passwd_rec_t *new_key,
        sec_passwd_type_t new_keytype,
        sec_passwd_version_t *new_key_version,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name

A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. All three strings must be completely specified.

caller_key

The key to use to encrypt the key for transmission to the registry server.

new_key The password for the new account. During transmission to the registry server, it is encrypted with *caller_key*.

new_keytype
The type of the new key. The server uses this parameter to decide how to encode *new_key* if it is sent as plaintext.

Output

new_key_version
The key version number returned by the server. If the client requests a particular key version number (via the *version_number* field of the *new_key* input parameter), the server returns the requested version number back to the client.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_passwd()** routine changes an account password to the input password character string. Wildcards (blank fields) are not permitted in the specified account name; the principal, group, and organization names of the account must be completely specified.

Permissions Required

The **sec_rgy_acct_passwd()** routine requires the **u** (**user_info**) permission on the account principal whose password is to be changed.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to change the password of this account.

sec_rgy_object_not_found
The account to be modified was not found by the registry server.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_acct_rename

Purpose

Changes an account login name

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_rename(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *old_login_name,
        sec_rgy_login_name_t *new_login_name,
        sec_rgy_acct_key_t *new_key_parts,
        error_status_t *status);
```

Parameters

Input

context
: An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

old_login_name
: A pointer to the current account login name. The login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. All three strings must be completely specified.

new_login_name
: A pointer to the new account login name. Again, all three component names must be completely specified.

Input/Output

new_key_parts
: A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_rename()** routine changes an account login name from *old_login_name* to *new_login_name*. Wildcards (empty fields) are not permitted in either input name; both the old and new login names must completely specify their component principal, group, and organization names. Note, though, that the principal component in a login name cannot be changed.

The *new_key_parts* variable identifies how many of the *new_login_name* parts to use as the unique abbreviation for the account. If the requested abbreviation duplicates an existing abbreviation for another account, the routine identifies the next shortest unique abbreviation and returns this abbreviation using *new_key_parts*.

Permissions Required

The **sec_rgy_acct_rename()** routine requires the **m** (**mgmt_info**) permission on the account principal to be renamed.

Notes

The **sec_rgy_acct_rename()** routine does not affect any of the registry PGO data. The constituent principal, group, and organization items for an account must be added before the account can be created. (See the **sec_rgy_pgo_add()** routine). Also, the principal must have been added as a member of the specified group and organization. (See the **sec_rgy_pgo_add_member()** routine).

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to make the changes.

sec_rgy_object_not_found
The account to be modified was not found by the registry server.

sec_rgy_name_exists
The new account name is already in use by another account.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**.

sec_rgy_acct_replace_all

Purpose

Replaces all account data for an account

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_replace_all(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_acct_user_t *user_part,
        sec_rgy_acct_admin_t *admin_part,
        boolean32 set_password,
        sec_passwd_rec_t *caller_key,
        sec_passwd_rec_t *new_key,
        sec_passwd_type_t new_keytype,
        sec_passwd_version_t *new_key_version,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name

A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. For the group and organization names, blank strings can serve as wildcards, matching any entry. The principal name must be input.

user_part A pointer to the **sec_rgy_acct_user_t** structure containing the user part of the account data. This information can be modified only by the account owner or other authorized user.

admin_part
A pointer to the **sec_rgy_acct_admin_t** structure containing the administrative part of an account's data. This information includes the account creation and expiration dates and flags describing limits to the use of privilege attribute certificates, among other information, and can be modified only by an administrator.

set_passwd The password reset flag. If you set this parameter to TRUE, the account's password will be changed to the value specified in *new_key*.

caller_key A key to use to encrypt the key for transmission to the registry server. If communications secure to the **rpc_c_authn_level_pkt_privacy** level are available on a system, then this parameter is not necessary, and the packet encryption is sufficient to ensure security.

new_key The password for the new account. During transmission to the registry server, it is encrypted with *caller_key*.

new_keytype
The type of the new key. The server uses this parameter to decide how to encode the plaintext key.

Input/Output

key_parts A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

Output

new_key_version
The key version number returned by the server. If the client requests a particular key version number (via the *version_number* field of the *new_key* input parameter), the server returns the requested version number back to the client.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_replace_all()** routine replaces both the user and administrative information in the account record specified by the input login name. The administrative information contains limitations on the account's use and privileges. The user information contains information such as the account home directory and default shell. The administrative information can only be modified by a registry administrator or another authorized user (users with **admin_info** (**a**) and **mgmt_info** (**m**) privileges for an account). The user information can be modified by the account owner or another authorized user (users with **user_info** (**u**) privileges for an account).

Use the *set_passwd* parameter to reset the account password. If you set this parameter to TRUE, the account's pasword is changed to the value specified in *new_key*.

The *key_parts* variable identifies how many of the *login_name* parts to use as the unique abbreviation for the replaced account. If the requested abbreviation duplicates an existing abbreviation for another account, the routine identifies the next shortest unique abbreviation and returns this abbreviation using *key_parts*.

Permissions Required

The **sec_rgy_acct_replace_all()** routine requires the following permissions on the account principal:

- The **m** (**mgmt_info**) permission, if **flags** or **expiration_date** is to be changed.
- The **a** (*auth_info*) permission, if **authentication_flags** or **good_since_date** is to be changed.
- The **u** (**user_info**) permission, if user **flags**, **gecos**, **homedir** (home directory), **shell**, or **passwd** (password) are to be changed.

Notes

All users need the **w** (**write**) privilege to modify any account information.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to change account information.

sec_rgy_object_not_found
The specified account could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_acct_admin_replace(3sec)**, **sec_rgy_acct_rename(3sec)**, **sec_rgy_acct_user_replace(3sec)**.

sec_rgy_acct_user_replace

Purpose

Replaces user account data

Synopsis

```
#include <dce/acct.h>

void sec_rgy_acct_user_replace(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_user_t *user_part,
        boolean32 set_passwd,
        sec_passwd_rec_t *caller_key,
        sec_passwd_rec_t *new_key,
        sec_passwd_type_t new_keytype,
        sec_passwd_version_t *new_key_version,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

login_name

A pointer to the account login name. A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. For the group and organization names, blank strings can serve as wildcards, matching any entry. The principal name must be input.

user_part A pointer to the **sec_rgy_acct_user_t** structure containing the user part of the account data. This information can be modified only by the account owner or other authorized user. The structure contains the following fields:

gecos A character string containing information about the account owner. This often includes such information as their name and telephone number.

homedir The default directory upon login for the account.

shell The default shell to use upon login.

passwd_version_number
The password version number, a 32-bit unsigned integer, set by the registry server.

passwd_dtm
The date and time of the last password change (in **sec_timeval_sec_t** form), also set by the registry server.

flags A flag set of type **sec_rgy_acct_user_flags_t**.

passwd The account's encrypted password.

The only user data fields that can be changed are: **gecos**, **homedir**, **shell**, **flags**, and **passwd**.

set_passwd
The password reset flag. If you set this parameter to TRUE, the user's password will be changed to the value specified in *new_key*.

caller_key A key to use to encrypt the key for transmission to the registry server. If communications secure to the **rpc_c_authn_level_pkt_privacy** level are available on a system, then this parameter is not necessary, and the packet encryption is sufficient to ensure security.

new_key The password for the new account. During transmission to the registry server, it is encrypted with *caller_key*.

new_keytype
The type of the new key. The server uses this parameter to decide how to encode the plaintext key.

Output

new_key_version
The key version number returned by the server. If the client requests a particular key version number (via the *version_number* field of the *new_key* input parameter), the server returns the requested version number back to the client.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_acct_user_replace()** routine replaces the user information in the account record specified by the input login name. The user information consists of information such as the account home directory and default shell. The the user information can be modified only by the account owner or other authorized users (users with **user_info** (**u**) privileges for an account).

Use the *set_passwd* parameter to reset the user's password. If you set this parameter to TRUE, the user's pasword is changed to the value specified in *new_key*.

Permissions Required

The **sec_rgy_acct_user_replace()** routine requires the **u** (**user_info**) permission on the account principal.

Notes

All users need the **w** (**write**) privilege to modify any account information.

Files

/usr/include/dce/acct.idl
The **idl** file from which **dce/acct.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to modify the account data.

sec_rgy_object_not_found
The specified account could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_acct_admin_replace(3sec)**, **sec_rgy_acct_rename(3sec)**, **sec_rgy_acct_replace_all(3sec)**.

sec_rgy_attr_cursor_alloc

Purpose

Allocates resources to a cursor used by **sec_rgy_attr_lookup_by_id**

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_cursor_alloc(
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Output

cursor A pointer to a **sec_attr_cursor_t**.

status A pointer to the completion status. On successful completion, the call returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_cursor_alloc()** call allocates resources to a cursor used with the **sec_rgy_attr_lookup_by_id** call. This routine, which is a local operation, does not initialize *cursor*.

The **sec_rgy_attr_cursor_init()** routine, which makes a remote call, allocates and initializes the cursor. In addition, **sec_rgy_attr_cursor_init()** returns the total number of attributes attached to the object as an output parameter; **sec_rgy_attr_cursor_alloc()** does not.

Permissions Required

None.

Files

/usr/include/dce/sec_attr_base.idl
The **idl** file from which **dce/sec_attr_base.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

no such object

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_cursor_init(3sec)**, **sec_rgy_attr_cursor_release(3sec)**, **sec_rgy_attr_cursor_reset(3sec)**, **sec_rgy_attr_lookup_by_id(3sec)**.

sec_rgy_attr_cursor_init

Purpose

Initializes a cursor used by **sec_rgy_attr_lookup_by_id**

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_cursor_init (
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        unsigned32 *cur_num_attrs,
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the registry domain in which the object specified by *name* resides. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A pointer to a **sec_rgy_name_t** character string containing the name of the person, group, or organization to which the attribute to be scanned is attached.

Output

cur_num_attrs
A pointer to an unsigned 32-bit integer that specifies the number of attributes currently attached to the object.

cursor A pointer to a **sec_rgy_cursor_t** positioned at the first attribute in the list of the object's attributes.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_cursor_init()** routine initializes a cursor of type **sec_attr_cursor_t** (used with the **sec_rgy_attr_lookup_by_id** call) and initializes the cursor to the first attribute in the specified object's list of attributes. This call also supplies the total number of attributes attached to the object as part of its output. The cursor allocation is a local operation. The cursor initialization is a remote operation and makes a remote call to the registry.

Use the **sec_rgy_attr_cursor_release()** call to release all resources allocated to a **sec_attr_cursor_t** cursor.

Permissions Required

The **sec_rgy_attr_cursor_init()** routine requires at least one permission (of any type) on the person, group, or organization to which the attribute to be scanned is attached.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

no such object

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_cursor_release,sec_rgy_attr_lookup_by_id**.

sec_rgy_attr_cursor_release

Purpose

Releases a cursor

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_cursor_release (
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

Input/Output

cursor As an input parameter, a pointer to an uninitialized cursor of type **sec_attr_cursor_t**. As an output parameter, a pointer to an uninitialized cursor of type **sec_attr_cursor_t** with all resources released.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_cursor_release()** routine releases all resources allocated to a **sec_attr_cursor_t** by the **sec_rgy_attr_cursor_init()** or **sec_rgy_attr_cursor_alloc()** call.

This is a local-only operation and makes not remote calls.

Permissions Required

None.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

No such object

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_cursor_alloc(3sec)**, **sec_rgy_attr_cursor_init(3sec)**, **sec_rgy_attr_lookup_by_id**.

sec_rgy_attr_cursor_reset

Purpose

Reinitializes a cursor

Synopsis

```
#include <dce/sec_attr_base.h>

void sec_attr_cursor_reset(
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As an input parameter, an initialized *cursor*. As an output parameter, *cursor* is reset to the first attribute in the schema.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_cursor_reset()** routine resets a **dce_attr_cursor_t** that has been allocated by either a **sec_rgy_attr_cursor_init()** or **sec_rgy_attr_cursor_alloc()**. The reset cursor can then be used to process a new **sec_rgy_attr_lookup_by_id** query by reusing the cursor instead of releasing and reallocating it. This is a local operation and makes no remote calls.

Permissions Required

None.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_cursor_alloc(3sec)**, **sec_rgy_attr_cursor_init(3sec)**, **sec_rgy_attr_lookup_by_id(3sec)**.

sec_rgy_attr_delete

Purpose

Deletes specified attributes for a specified object

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_delete (
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        unsigned32 num_to_delete,
        sec_attr_t attrs[ ],
        signed32 *failure_index,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the registry domain in which the object identified by *name* resides. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A character string of type **sec_rgy_name_t** specifying the name of the person, group, or organization to which the attributes are attached.

num_to_delete
A 32-bit integer that specifies the number of elements in the *attrs[ ]* array. This integer must be greater than 0.

attrs[ ] An array of values of type **sec_attr_t** that specifies the attribute instances to be deleted. The size of the array is determined by *num_to_delete*.

Output

failure_index
In the event of an error, *failure_index* is a pointer to the element in the *in_attrs[ ]* array that caused the update to fail. If the failure cannot be attributed to a specific attribute, the value of *failure_index* is **–1**.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_delete()** routine deletes attributes. This is an atomic operation: if the deletion of any attribute in the *attrs[ ]* array fails, all deletions are aborted. The attribute causing the delete to fail is identified in *failure_index*. If the failure cannot be attributed to a given attribute, *failure_index* contains **-1**.

The *attrs[ ]* array, which specifies the attributes to be deleted, contains values of type **sec_attr_t** These values consist of

- *attr_id*, a UUID that identifies the attribute type
- *attr_value*, values of **sec_attr_value_t** that specify the attribute's encoding type and values.

To delete attributes that are not multivalued and to delete all instances of a multivalued attribute, an attribute UUID is all that is required. For these attribute instances, supply the attribute UUID in the input array and set the attribute encoding (in **sec_attr_encoding_t**) to **sec_attr_enc_void**.

To delete a specific instance of a multivalued attribute, supply the UUID and value that uniquely identify the multivalued attribute instance in the input array.

Note that if the deletion of any attribute instance in the array fails, all fail. However, to help pinpoint the cause of the failure, the call identifies the first attribute whose deletion failed in a failure index by array element number.

Permissions Required

The **sec_rgy_attr_delete()** routine requires the delete permission set for each attribute type identified in the *attrs[ ]* array. These permissions are defined as part of the ACL manager set in the schema entry for the attribute type.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

database read only

invalid/unsupported attribute type

server unavailable

site read only

unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_update(3sec)**.

sec_rgy_attr_get_effective

Purpose

Reads effective attributes by ID

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_get_effective(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        unsigned32 num_attr_keys,
        sec_attr_t attr_keys[ ],
        sec_attr_vec_t *attr_list,
        error_status_t status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the domain in which the named object resides. The valid values are as follows:

sec_rgy_domain_principal
The *name* identifies a principal.

sec_rgy_domain_group
The *name* identifies a group.

sec_rgy_domain_org
The *name* identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A pointer to a **sec_rgy_name_t** character string containing the name of the person, group, or organization to which the attribute is attached.

num_attr_keys
An unsigned 32-bit integer that specifies the number of elements in the the *attr_keys*[ ] array. If *num_attr_keys* is set to 0 (zero), all of the effective attributes that the caller is authorized to see are returned.

attr_keys[ ] An array of values of type **sec_attr_t** that specify the UUIDs of the attributes to be returned if they are effective. If the attribute type is associated with a query attribute trigger, the **sec_attr_t** *attr_value* field can be used to pass in optional information required by the attribute trigger query. If no information is to be passed in the *attr_value* field (whether the type indicates an attribute trigger query or not), set the attribute's encoding type to **sec_rgy_attr_enc_void**. The size of the *attr_keys*[ ] array is determined by the *num_attr_keys* parameter.

Output

attr_list A pointer an attribute vector allocated by the server containing all of the effective attributes matching the search criteria (defined in *num_attr_keys* or *attr_keys*[ ]). The server allocates a buffer large enough to return all the requested attributes so that subsequent calls are not necessary.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_get_effective()** routine returns the UUIDs of a specified object's effective attributes. Effective attributes are determined by setting of the schema entry **apply_defaults** flag:

- If the flag is set off, only the attributes directly attached to the object are effective.
- If the flag is set on, the effective attributes are obtained by performing the following steps for each attribute identified by UUID in the *attr_keys* array:
  - — If the object named by *name* is a principal and if the a requested attribute exists on the principal, that attribute is effective and is returned. If it does not exist, the search continues.
  - — The next step in the search depends on the type of object:

For principals with accounts:

The organization named in the principal's account is examined to see if an attribute of the requested type exists. If it does, it is effective and is returned; then the search for that attribute ends. If it does not exist, the search for that attribute continues to the **policy** object as described here.

The registry **policy** object is examined to see if an attribute of the requested type exits. If it does, it is returned. If it does not, a message indicating the no attribute of the type exists for the object is returned.

For principals without accounts, for groups, and for organizations:

The registry **policy** object is examined to see if an attribute of the requested type exits. If it does, it is returned. If it does not, a message indicating the no attribute of the type exists for the object is returned.

For multivalued attributes, the call returns a **sec_attr_t** for each value as an individual attribute instance. For attribute sets, the call returns a **sec_attr_t** for each member of the set; it does not return the set instance.

If the attribute instance to be read is associated with a query attribute trigger that requires additional information before it can process the query request, use a **sec_attr_value_t** to supply the requested information. To do this

- Set the **sec_attr_encoding_t** to an encoding type that is compatible with the information required by the query attribute trigger.
- Set the **sec_attr_value_t** to hold the required information.

If the attribute instance to be read is not associated with a query trigger or no additional information is required by the query trigger, an attribute UUID is all that is required. For these attribute instances, supply the attribute UUID in the input array and set the attribute encoding (in **sec_attr_encoding_t**) to **sec_attr_enc_void**.

If the requested attribute type is associated with a query trigger, the value returned for the attribute will be the binding (as set in the schema entry) of the trigger server. The caller must bind to the trigger server and pass the original input query attribute to the **sec_attr_trig_query** call in order to retrieve the attribute value.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_attr_lookup_by_id

Purpose

Reads a specified object's attributes, expanding attribute sets into individual member attributes

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_lookup_by_id (
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        sec_attr_cursor_t *cursor,
        unsigned32 num_attr_keys,
        unsigned32 space_avail,
        sec_attr_t attr_keys[ ],
        unsigned32 *num_returned,
        sec_attr_t attrs[ ],
        unsigned32 *num_left,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the registry domain in which the object specified by *name* resides. The valid values are as follows:

sec_rgy_domain_person

The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A pointer to a **sec_rgy_name_t** character string containing the name of the person, group, or organization to which the attribute is attached.

num_attr_keys
An unsigned 32-bit integer that specifies the number of elements in the *attr_keys[ ]* array. Set this parameter to 0 (zero) to return all of the object's attributes that the caller is authorized to see.

space_avail
An unsigned 32-bit integer that specifies the size of the *attr_keys[ ]* array.

attr_keys[ ] An array of values of type **sec_attr_t** that identify the attribute type ID of the attribute instance(s) to be looked up. If the attribute type is associated with a query attribute trigger, the **sec_attr_t** *attr_value* field can be used to pass in optional information required by the attribute trigger query. If no information is to be passed in the *attr_value* field (whether the type indicates an attribute trigger query or not), set the attribute's encoding type to **sec_rgy_attr_enc_void**.

The size of the *attr_keys[ ]* array is determined by the *num_attr_keys* parameter.

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As an input parameter, *cursor* is a pointer to a **sec_attr_cursor_t** initialized by a **sec_rgy_attr_srch_cursor_init** call. As an output parameter, *cursor* is a pointer to a **sec_attr_cursor_t** that is positioned past components returned in this call.

Output

num_returned
A pointer to a 32-bit unsigned integer that specifies the number of attribute instances returned in the *attrs[ ] array*.

attrs[ ] An array of values of type **sec_attr_t** that contains the attributes retrieved by Universal Unique Identifier (UUID). The size of the array is determined by *space_avail* and the length by *num_returned*.

num_left A pointer to a 32-bit unsigned integer that supplies the number of attributes that were found but could not be returned because of space constraints in the *attrs[ ]* buffer. To ensure that all the attributes will be returned, increase the size of the *attrs[ ]* array by increasing the size of *space_avail* and *num_returned*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**, or, if the requested attributes were not available, it returns the message **not_all_available**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_lookup_by_id()** function reads those attributes specified by UUID for an object specified by name. This routine is similar to the **sec_rgy_attr_lookup_no_expand()** routine with one exception: for attribute sets, the **sec_rgy_attr_lookup_no_expand()** routine returns a **sec_attr_t** for the set instance only; it does not expand the set and return a **sec_attr_t** for each member in the set. This call expands attribute sets and returns a **sec_attr_t** for each member in the set.

If the *num_attr_keys* parameter is set to 0 (zero), all of the object's attributes that the caller is authorized to see are returned. This routine is useful for programmatic access.

After a successful call, free the resources allocated by this routine for each attribute returned in the *attrs[ ]* parameter with the **sec_attr_util_inst_free_ptrs()** routine.

For multivalued attributes, the call returns a **sec_attr_t** for each value as an individual attribute instance. For attribute sets, the call returns a **sec_attr_t** for each member of the set; it does not return the set instance.

The *attr_keys[ ]* array, which specifies the attributes to be returned, contains values of type **sec_attr_t**. These values consist of the following:

- *attr_id*, a UUID that identifies the attribute type
- *attr_value*, values of **sec_attr_value_t** that specify the attribute's encoding type and values.

Use the *attr_id* field of each *attr_keys[ ]* array element, to specify the UUID that identifies the attribute type to be returned.

If the attribute instance to be read is not associated with a query trigger or no additional information is required by the query trigger, an attribute UUID is all that is required. For these attribute instances, supply the attribute UUID in the input array and set the attribute encoding (in **sec_attr_encoding_t**) to **sec_attr_enc_void**.

If the attribute instance to be read is associated with a query attribute trigger that requires additional information before it can process the query request, use a **sec_attr_value_t** to supply the requested information, as follows:

1. Set the **sec_attr_encoding_t** to an encoding type that is compatible with the information required by the query attribute trigger.
2. Set the **sec_attr_value_t** to hold the required information.

Note that if you set *num_attr_keys* to zero to return all of the object's attributes and that attribute is associated with a query attribute trigger, the attribute trigger will be called with no input attribute information (that would normally have been passed in via the *attr_value* field).

The *cursor* parameter specifies a cursor of type **sec_attr_cursor_t** initialized to the point in the attribute list at which to start processing the query. Use the **sec_attr_cursor_init** function to initialize *cursor*. If *cursor* is uninitialized, the behavior is undefined.

The *num_left* parameter contains the number of attributes that were found but could not be returned because of space constraints in the *attrs[ ]* array. (Note that this number may be inaccurate if the target server allows updates between successive queries.) To obtain all of the remaining attributes, set the size of the *attrs[ ]* array so that it is large enough to hold the number of attributes listed in *num_left*.

Permissions Required

The **sec_rgy_attr_lookup_by_id()** routine requires the **q** (**query**) permission set for each attribute type identified in the *attr_keys[ ]* array. These permissions are defined as part of the access control list (ACL) manager set in the schema entry of each attribute type.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

unauthorized

registry server unavailable

trigger server unavailable

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_attr_lookup_by_name(3sec)**, **sec_rgy_attr_lookup_no_expand(3sec)**.

sec_rgy_attr_lookup_by_name

Purpose

Reads a single attribute instance for a specific object

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_lookup_by_name(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        char *attr_name,
        sec_attr_t *attr,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the domain in which the named object resides. The valid values are as follows:

sec_rgy_domain_principal
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A pointer to a **sec_rgy_name_t** character string containing the name of the person, group, or organization to which the attribute is attached.

attr_name A pointer to a character string that specifies the name of the attribute to be retrieved.

Output

attr A pointer to a **sec_attr_t** that contains the first instance of the named attribute.

status A pointer to the completion status. The completion status can be one of the following:

error_status_ok
All instances of the value were returned with no errors.

more_available
A multivalued attribute was specified as *name* and the routine completed successfully. For multivalued attributes, this routine returns the first instance of the attribute.

attribute_set_instance
An attribute set was specified as *name* and the routine completed successfully.

error message
An error message if the routine did not complete successfully.

Description

The **sec_rgy_attr_lookup_by_name()** routine returns the named attribute for a named object. This routine is useful for an interactive editor.

For multivalued attributes, this routine returns the first instance of the attribute. To retrieve every instance of the attribute, use the **sec_rgy_attr_lookup_by_id** call, supplying the attribute Universal Unique Identifier (UUID) returned in the *attr* parameter.

For attribute sets, the routine returns the attribute set instance, not the member instances. To retrieve all members of the set, use the **sec_rgy_attr_lookup_by_id** call, supplying the the attribute set UUID returned in the *attr* parameter.

After a successful call, free the resources allocated by this routine for the **attr** parameter, with the **sec_attr_util_inst_free_ptrs()** routine.

This routine does not provide for input data to an attribute trigger query operation. If the named attribute is associated with a query attribute trigger, the attribute trigger will be called with no input attribute value information.

Permissions Required

The **sec_rgy_attr_lookup_by_name()** routine requires the **q** (**query**) permission set for the attribute type of the attribute instance identified by *attr_name*. These permissions are defined as part of the access control list (ACL) manager set in the schema entry of each attribute type.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

registry server unavailable

trigger server unavailable

unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_lookup_by_id(3sec)**, **sec_rgy_attr_lookup_no_expand(3sec)**.

sec_rgy_attr_lookup_no_expand

Purpose

Reads a specified object's attribute(s), without expanding attribute sets into individual member attributes

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_lookup_no_expand(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        sec_attr_cursor_t *cursor,
        unsigned32 num_attr_keys,
        unsigned32 space_avail,
        uuid_t attr_keys[ ],
        unsigned32 *num_returned,
        sec_attr_t attr_sets[ ],
        unsigned32 *num_left,
        error_status_t status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

A value of type **sec_rgy_domain_t** that identifies the domain in which the named object resides. The valid values are as follows:

sec_rgy_domain_principal
The *name* identifies a principal.

sec_rgy_domain_group
The *name* identifies a group.

sec_rgy_domain_org
The *name* identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A pointer to a **sec_rgy_name_t** character string containing the name of the person, group, or organization to which the attribute is attached.

num_attr_keys
An unsigned 32-bit integer that specifies the number of elements in the the *attr_keys[ ]* array. If *num_attr_keys* is set to 0 (zero), all attribute sets that the caller is authorized to see are returned.

space_avail
An unsigned 32-bit integer that specifies the size of the *attrs_sets*[ ] array.

attr_keys[ ] An array of values of type **uuid_t** that specify the UUIDs of the attribute sets to be returned. The size of the *attr_keys[ ]* array is determined by the *num_attr_keys* parameter.

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As an input parameter, *cursor* is a pointer to a **sec_attr_cursor_t** that is initialized by the **sec_rgy_attr_cursor_init**. As an output parameter, *cursor* is a pointer to a **sec_attr_cursor_t** that is positioned past the attribute sets returned in this call.

Output

num_returned
A pointer to a 32-bit integer that specifies the number of attribute sets returned in the *attrs*[ ] array.

attr_sets[ ] An array of values of type **sec_attr_t** that contains the attribute sets retrieved by UUID. The size of the array is determined by *space_avail* and the length by *num_returned*.

num_left A pointer to a 32-bit unsigned integer that supplies the number of attribute sets that were found but could not be returned because of space constraints in the *attr_sets[ ]* buffer. To ensure that all the attributes will be returned, increase the size of the *attr_sets[ ]* array by increasing the size of *space_avail* and *num_returned*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_lookup_no_expand()** routine reads attribute sets. This routine is similar to the **sec_rgy_attr_lookup_by_id()** routine with one exception: for attribute sets, the **sec_rgy_attr_lookup_by_id()** routine expands attribute sets and returns a **sec_attr_t** for each member in the set. This call does not. Instead it returns a **sec_attr_t** for the set instance only. The **sec_rgy_attr_lookup_no_expand()** routine is useful for programmatic access.

cursor is a cursor of type **sec_attr_cursor_t** that establishes the point in the attribute set list from which the server should start processing the query. Use the **sec_rgy_attr_cursor_init** function to initialize *cursor*. If *cursor* is uninitialized, the behavior is undefined.

The *num_left* parameter contains the number of attribute sets that were found but could not be returned because of space constraints of the *attr_sets[ ]* array. (Note that this number may be inaccurate if the target server allows updates between successive queries.) To obtain all of the remaining attribute sets, set the size of the *attr_sets[ ]* array so that it is large enough to hold the number of attributes listed in *num_left*.

Permissions Required

The **sec_rgy_attr_lookup_no_expand()** routine requires the query permission set for each attribute type identified in the *attr_keys[ ]* array. These permissions are defined as part of the ACL manager set in the schema entry of each attribute type.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

invalid/unsupported attribute type

registry server unavailable

unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_lookup_by_id(3sec)**, **sec_rgy_attr_lookup_by_name(3sec)**.

sec_rgy_attr_sch_aclmgr_strings

Purpose

Returns printable ACL strings associated with an ACL manager protecting a schema object

Synopsis

```
#include  <dce/dce_attr_base.h>

void sec_rgy_attr_sch_aclmgr_strings(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        uuid_t *acl_mgr_type,
        unsigned32 size_avail,
        uuid_t *acl_mgr_type_chain,
        sec_acl_printstring_t *acl_mgr_info,
        boolean32 *tokenize,
        unsigned32 *total_num_printstrings,
        unsigned32 *size_used,
        sec_acl_printstring_t permstrings[ ],
        error_status_t *status);
```

Parameters

Input

context
An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

acl_manager_type
A pointer to the UUID identifying the type of the ACL manager in question. There may be more than one type of ACL manager protecting the schema object whose ACL is bound to the input handle. Use this

parameter to distinguish them. Use **sec_rgy_attr_sch_get_acl_mgrs()** to acquire a list of the manager types protecting a given schema object.

size_avail An unsigned 32-bit integer containing the allocated length of the *permstrings[ ]* array.

Output

acl_mgr_type_chain
If the target object ACL contains more than 32 permission bits, chains of manager types are used: each manager type holds one 32-bit segment of permissions. The UUID returned in *acl_mgr_type_chain* refers to the next ACL manager in the chain. If there are no more ACL managers in the chain, **uuid_nil** is returned.

acl_mgr_info
A pointer to a printstring that contains the ACL manager type's name, help information, and set of supported of permission bits.

tokenize A pointer to a variable that specifies whether or not printstrings will be passed separately:

- TRUE indicates that the printstrings must be printed or passed separately.
- FALSE indicates that the printstrings are unambiguous and can be concatenated when printed without confusion.

total_num_printstrings
A pointer to an unsigned 32-bit integer containing the total number of permission entries supported by this ACL manager type.

size_used A pointer to an unsigned 32-bit integer containing the number of permission entries returned in the *permstrings[ ]* array.

permstrings[ ]
An array of printstrings of type **sec_acl_printstring_t**. Each entry of the array is a structure containing the following three components:

printstring A character string of maximum length **sec_acl_printstring_len** describing the printable representation of a specified permission.

helpstring A character string of maximum length **sec_acl_printstring_help_len** containing some text that can be used to describe the specified permission.

permissions A **sec_acl_permset_t** permission set describing the permissions that are represented with the companion printstring.

The array consists of one such entry for each permission supported by the ACL manager identified by *acl_mgr_type*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_aclmgr_strings()** routine returns an array of printable representations (called *printstrings*) for each permission bit or combination of permission bits the specified ACL manager supports. The ACL manager type specified by *acl_mgr_type* must be one of the types protecting the schema object bound to by *h*.

In addition to returning the printstrings, this routine also returns instructions about how to print the strings in the *tokenize* variable. If this variable is set to FALSE, the printstrings can be concatenated. If it is set to TRUE, the printstrings cannot be concatenated. For example a printstrings of **r** or **w** could be concatenated as **rw** without any confusion. However, printstrings in a form of **read** or **write**, should not be concatenated.

ACL managers often define aliases for common permission combinations. By convention, simple entries appear at the beginning of the *printstrings*[ ] array, and combinations appear at the end.

Permissions Required

The **sec_rgy_attr_sch_scl_mgr_strings()** routine requires the **r** permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_no_memory

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_get_acl_mgrs(3sec)**.

sec_rgy_attr_sch_create_entry

Purpose

Creates a schema entry

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_create_entry(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        sec_attr_schema_entry_t *schema_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

schema_entry
A pointer to a **sec_attr_schema_entry_t** that contains the schema entry values for the schema in which the entry is to be created.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_create_entry()** routine creates schema entries that define attribute types.

Permissions Required

The **sec_rgy_attr_sch_create_entry()** routine requires **i** permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_bad_acl_mgr_set

sec_attr_bad_acl_mgr_type

sec_attr_bad_bind_authn_svc

sec_attr_bad_bind_authz_svc

sec_attr_bad_bind_info

sec_attr_bad_bind_prot_level

sec_attr_bad_bind_svr_name

sec_attr_bad_comment

sec_attr_bad_encoding_type

sec_attr_bad_intercell_action

sec_attr_bad_name

sec_attr_bad_permset

sec_attr_bad_scope

sec_attr_bad_uniq_query_accept

sec_attr_name_exists

sec_attr_no_memory

sec_attr_svr_read_only

sec_attr_svr_unavailable

sec_attr_trig_bind_info_missing

sec_attr_type_id_exists

sec_attr_unauthorized

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_delete_entry(3sec)**, **sec_rgy_attr_sch_update(3sec)**.

sec_rgy_attr_sch_cursor_alloc

Purpose

Allocates resources to a cursor used with **sec_rgy_attr_sch_scan**

Synopsis

```
void sec_rgy_att_sch_cursor_alloc(
        dce_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Output

cursor A pointer to a **sec_attr_cursor_t**.

status A pointer to the completion status. On successful completion, the call returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_cursor_alloc()** call allocates resources to a cursor used with the **sec_rgy_attr_sch_scan()** call. This routine, which is a local operation, does not initialize *cursor*.

The **sec_rgy_attr_sch_cursor_init()** routine, which makes a remote call, allocates and initializes the cursor. In addition, **sec_rgy_attr_sch_cursor_init() returns the total number of entries found in the schema as an output parameter; sec_rgy_attr_sch_cursor_alloc()** does not.

Permissions Required

None.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.id** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_no_memory

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_cursor_init(3sec)**, **sec_rgy_attr_sch_cursor_release(3sec)**, **sec_rgy_attr_sch_scan(3sec).**

sec_rgy_attr_sch_cursor_init

Purpose

Initializes and allocates a cursor used with **sec_rgy_attr_sch_scan**

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_cursor_init(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        unsigned32 *cur_num_entries,
        sec_attr_cursor_t *cursor,
        error_status_t status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

Output

cur_num_entries
A pointer to an unsigned 32-bit integer that specifies the total number of entries contained in the schema at the time of this call.

cursor A pointer to a **sec_attr_cursor_t** that is initialized to the first entry in the the schema.

status A pointer to the completion status. On successful completion, the call returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_cursor_init()** call initializes and allocates resources to a cursor used with the **sec_rgy_attr_sch_scan** call. This call makes remote calls to initialize the cursor.

To limit the number of remote calls, use the **sec_rgy_attr_sch_cursor_alloc()** call to allocate *cursor*, but not initialize it. Be aware, however, that the the **sec_rgy_attr_sch_cursor_init()** call supplies the total number of entries found in the schema as an output parameter; the **sec_rgy_attr_sch_cursor_alloc()** call does not.

If the cursor iunput to **sec_rgy_attr_sch_scan** has not been initialized, the **sec_rgy_attr_sch_scan** call will initialize it; if it has been initialized, **sec_rgy_attr_sch_scan** advances it.

Permissions Required

None.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_no_memory

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_cursor_alloc(3sec)**,**sec_rgy_attr_sch_cursor_release(3sec)**, **sec_rgy_attr_sch_scan(3sec).**

sec_rgy_attr_sch_cursor_release

Purpose

Releases states associated with a cursor used by **sec_rgy_attr_sch_scan**

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_cursor_release(
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As an input parameter, *cursor* must have been initialized to the first entry in a schema. As an output parameter, *cursor* is uninitialized with all resources releases.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_cursor_init()** routine releases the resources allocated to the cursor used by the **sec_rgy_attr_sch_scan** routine. This call is a local operation and makes no remote calls.

Permissions Required

None.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_cursor_allocate(3sec)**, **sec_rgy_attr_sch_cursor_init(3sec)**, **sec_rgy_attr_sch_scan(3sec)**.

sec_rgy_attr_sch_cursor_reset

Purpose

Resets a cursor that has been allocated

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void dce_attr_cursor_reset(
        sec_attr_cursor_t *cursor,
        error_status_t *status);
```

Parameters

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As an input parameter, an initialized *cursor*. As an output parameter, *cursor* is reset to the first attribute in the schema.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_cursor_reset()** routine resets a **dce_attr_cursor_t** that has been allocated by either a **sec_rgy_attr_sch_cursor_init()** or **sec_rgy_attr_sch_cursor_alloc()**. The reset cursor can then be used to process a new **sec_rgy_attr_sch_scan** query by reusing the cursor instead of releasing and reallocating it. This is a local operation and makes no remote calls.

Permissions Required

None.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_bad_cursor

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_cursor_alloc(3sec)**, **sec_rgy_attr_sch_cursor_init(3sec)**, **sec_rgy_attr_sch_scan(3sec)**.

sec_rgy_attr_sch_delete_entry

Purpose

Deletes a schema entry

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_delete_entry(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        uuid_t *attr_id,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

attr_id A pointer to a **uuid_t** that identifies the schema entry to be deleted.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_delete_entry()** routine deletes a schema entry. Because this is a radical operation that invalidates any existing attributes of this type on objects dominated by the schema, access to this operation should be severely limited.

Permissions Required

The **sec_rgy_attr_sch_delete_entry()** routine requires the **d** permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_no_memory

sec_attr_sch_entry_not_found

sec_attr_svr_read_only

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_create_entry(3sec)**, **sec_rgy_attr_sch_update_entry(3sec)**.

sec_rgy_attr_sch_get_acl_mgrs

Purpose

Retrieves the manager types of the ACLs protecting the objects dominated by a named schema

Synopsis

```
#include  <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_get_acl_mgrs(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        unsigned32 size_avail,
        unsigned32 *size_used,
        unsigned32 *num_acl_mgr_types,
        uuid_t acl_mgr_types[ ],
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

size_avail An unsigned 32-bit integer containing the allocated length of the *acl_manager_types*[ ] array.

Output

size_used An unsigned 32-bit integer containing the number of output entries returned in the *acl_mgr_types[ ]* array.

num_acl_mgr_types
An unsigned 32-bit integer containing the number of types returned in the *acl_mgr_types[ ]* array. This may be greater than *size_used* if there

was not enough space allocated by *size_avail* for all the manager types in the *acl_manager_types*[ ] array.

acl_mgr_types[ ]
An array of the length specified in *size_avail* to contain UUIDs (of type **uuid_t**) identifying the types of ACL managers protecting the target object.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_get_acl_mgrs()** routine returns a list of the manager types protecting the schema object identified by *context*.

ACL editors and browsers can use this operation to determine the ACL manager types protecting a selected schema object. Then, using the **sec_rgy_attr_sch_aclmgr_strings()** routine, they can determine how to format for display the permissions supported by that ACL manager type.

Permissions Required

The **sec_rgy_attr_sch_get_acl_mgrs()** routine requires the **r** permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_no_memory

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_aclmgr_strings(3sec)**.

sec_rgy_attr_sch_lookup_by_id

Purpose

Reads a schema entry identified by UUID

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_lookup_by_id(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        uuid_t *attr_id,
        sec_attr_schema_entry_t *schema_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

attr_id A pointer to a **uuid_t** that identifies a schema entry.

Output

schema_entry
A **sec_attr_schema_entry_t** that contains an entry identified by *attr_id*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_lookup_by_id()** routine reads a schema entry identified by *attr_id*. This routine is useful for programmatic access.

After a successful call, use the **sec_attr_util_sch_ent_free_ptrs()** routine to free the resources allocated by this routine for the *schema_entry* parameter.

Permissions Required

The **sec_rgy_attr_sch_lookup_by_id()** routine requires the **r** (**read**) permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_sch_entry_not_found

sec_attr_svr_unavailable

sec_attr_unauthorized

sec_attr_no_memory

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_lookup_by_name(3sec)**, **sec_rgy_attr_sch_scan(3sec)**.

sec_rgy_attr_sch_lookup_by_name

Purpose

Reads a schema entry identified by name

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_lookup_by_name(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        char *attr_name,
        sec_attr_schema_entry_t *schema_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

attr_name A pointer to a character string that identifies the schema entry.

Output

schema_entry
A **sec_attr_schema_entry_t** that contains the schema entry identified by *attr_name*.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_lookup_by_name()** routine reads a schema entry identified by name. This routine is useful for use with an interactive editor.

After a successful call, use the **sec_attr_util_sch_ent_free_ptrs()** routine to free the resources allocated by this routine for the *schema_entry* parameter.

Permissions Required

The **sec_rgy_attr_sch_lookup_by_name()** routine requires the **r** (**read**) permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_bad_name

sec_attr_no_memory

sec_attr_sch_entry_not_found

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_lookup_by_id(3sec)**, **sec_rgy_attr_sch_scan(3sec)**.

sec_rgy_attr_sch_scan

Purpose

Reads a specified number of schema entries

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_scan(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        sec_attr_cursor_t *cursor,
        unsigned32 num_to_read,
        unsigned32 *num_read,
        sec_attr_schema_entry_t schema_entries[ ],
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

num_to_read
An unsigned 32-bit integer specifying the size of the *schema_entries[ ]* array and the maximum number of entries to be returned.

Input/Output

cursor A pointer to a **sec_attr_cursor_t**. As input *cursor* must be allocated and can be initialized. If *cursor* is not initialized, **sec_rgy_attr_sch_scan** will initialized. As output, *cursor* is positioned at the first schema entry after the returned entries.

Output

num_read A pointer an unsigned 32-bit integer specifying the number of entries returned in *schema_entries[ ]*.

schema_entries[ ]
A **sec_attr_schema_entry_t** that contains an array of the returned schema entries.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_scan()** routine reads schema entries. The read begins at the entry at which the input *cursor* is positioned and ends after the number of entries specified in *num_to_read*.

The input *cursor* must have been allocated by either the **sec_rgy_attr_sch_cursor_init()** or the **sec_rgy_attr_sch_cursor_alloc()** call. If the input *cursor* is not initialzed, **sec_rgy_attr_sch_scan()** initializes it; if cursor is initialized, **sec_rgy_attr_sch_scan()** simply advances it.

To read all entries in a schema, make successive **sec_rgy_attr_sch_scan()** calls. When all entries have been read, the call returns the message **no_more_entries**.

This routine is useful as a browser.

Permissions Required

The **sec_rgy_attr_sch_scan()** routine requires **r** permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_bad_cursor

sec_attr_no_memory

sec_attr_svr_unavailable

sec_attr_unauthorized

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_cursor_alloc(3sec)**, **sec_rgy_attr_sch_cursor_init(3sec)**, **sec_rgy_attr_sch_cursor_release(3sec)**.

sec_rgy_attr_sch_update_entry

Purpose

Updates a schema entry

Synopsis

```
#include <dce/sec_rgy_attr_sch.h>

void sec_rgy_attr_sch_update_entry(
        sec_rgy_handle_t context,
        sec_attr_component_name_t schema_name,
        sec_attr_schema_entry_parts_t modify_parts,
        sec_attr_schema_entry_t *schema_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

schema_name
Reserved for future use.

modify_parts
A value of type **sec_attr_schema_entry_parts_t** that identifies the fields in *schema_entry* that can be modified.

schema_entry
A pointer to a **sec_attr_schema_entry_t** that contains the schema entry values for the schema entry to be updated.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_sch_update_entry()** routine modifies schema entries. Only those schema entry fields set to be modified in the **sec_attr_schema_entry_parts_t** data type can be modified.

Some schema entry components can never be modified. Instead to make any changes to these components, the schema entry must be deleted (which deletes all attribute instances of that type) and recreated.

The schema entry components that can never be modified are as follows:

- Attribute name
- Reserved flag
- Apply defaults flag
- Intercell action flag
- Trigger types
- Comment

Fields that are arrays of structures (such as **acl_mgr_set** and **trig_binding**) are completely replaced by the new input array. This operation cannot be used to add a new element to the existing array.

Permissions Required

The **sec_rgy_attr_sch_update_entry()** routine requires the **M** (**Member_list**) permission on the **attr_schema** object.

Files

/usr/include/dce/sec_rgy_attr_sch.idl
The **idl** file from which **dce/sec_rgy_attr_sch.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_attr_field_no_update

sec_attr_bad_name

sec_attr_bad_acl_mgr_set

sec_attr_bad_acl_mgr_type

sec_attr_bad_permset

sec_attr_bad_intercell_action

sec_attr_trig_bind_info_missing

sec_attr_bad_bind_info

sec_attr_bad_bind_svr_name

sec_attr_bad_bind_prot_level

sec_attr_bad_bind_authn_svc

sec_attr_bad_bind_authz_svc

sec_attr_bad_uniq_query_accept

sec_attr_bad_comment

sec_attr_name_exists

sec_attr_sch_entry_not_found

sec_attr_unauthorized

sec_attr_svr_read_only

sec_attr_svr_unavailable

sec_attr_no_memory

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_sch_create_entry(3sec)**, **sec_rgy_attr_sch_delete_entry(3sec)**.

sec_rgy_attr_test_and_update

Purpose

Updates specified attribute instances for a specified object only if a set of control attribute instances match the object's existing attribute instances

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_test_and_update (
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        unsigned32 num_to_test,
        sec_attr_t test_attrs[ ],
        unsigned32 num_to_write,
        sec_attr_t update_attrs[ ],
        signed32 *failure_index,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
A value of type **sec_rgy_domain_t** that identifies the registry domain in which the object specified by *name* resides. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A character string of type **sec_rgy_name_t** specifying the name of the person, group, or organization to which the attribute is attached.

num_to_test
An unsigned 32-bit integer that specifies the number of elements in the *test_attrs[ ]* array. This integer must be greater than 0 (zero).

test_attrs[ ]
An array of values of type **sec_attr_t** that specifies the control attributes. The update takes place only if the types and values of the control attributes exactly match those of the attribute instances on the named registry object. The size of the array is determined by *num_to_test*.

num_to_write
A 32-bit integer that specifies the number of attribute instances returned in the *update_attrs[ ]* array.

update_attrs[ ]
An array of values of type **sec_attr_t** that specifies the attribute instances to be updated. The size of the array is determined by *num_to_write*.

Output

failure_index
In the event of an error, *failure_index* is a pointer to the element in the *update_attrs[ ]* array that caused the update to fail. If the failure cannot be attributed to a specific attribute, the value of *failure_index* is **−1**.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_test_and_update()** routine updates an attribute only if the set of control attributes specified in the *test_attrs[ ]* match attributes that already exist for the object.

This update is an atomic operation: if any of the control attributes do not match existing attributes, none of the updates are performed, and if an update should be performed,

but the write cannot occur for whatever reason to any member of the *update_attrs*[ ] array, all updates are aborted. The attribute causing the update to fail is identified in *failure_index*. If the failure cannot be attributed to a given attribute, *failure_index* contains **-1**.

If an attribute instance already exists which is identical in both *attr_id* and *attr_value* to an attribute specified in *in_attrs[ ]*, the existing attribute information is overwritten by the new information. For multivalued attributes, every instance with the same *attr_id* is overwritten with the supplied values.

If an attribute instance does not exist, it is created.

If you specify an attribute set for updating, the update applies to the set instance, the set itself, not the members of the set. To update a member of an attribute set, supply the UUID of the set member.

If an input attribute is associated with an update attribute trigger server, the attribute trigger server is invoked (by the **sec_attr_trig_update()** function) and the *in_attr*[ ] array is supplied as input. The output attributes from the update attribute trigger server are stored in the registry database and returned in the *out_attrs[ ]* array. Note that the update attribute trigger server may modify the values before they are used to update the registry database. This is the only circumstance under which the values in the *out_attrs[ ]* array differ from the values in the *in_attrs[ ]* array.

Permissions Required

The **sec_rgy_attr_test_and_update()** routine routine requires the test permission and the update permission set set for each attribute type identified in the *test_attrs[ ]* array. These permissions are defined as part of the ACL manager set in the schema entry of each attribute type.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

control attribute has changed

database read only

invalid encoding type

invalid/unsupported attribute type

server unavailable

site read only

trigger server unavailable

unauthorized

value not unique

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_delete(3sec)**, **sec_rgy_attr_update(3sec)**.

sec_rgy_attr_update

Purpose

Creates and updates attribute instances for a specified object

Synopsis

```
#include <dce/sec_rgy_attr.h>

void sec_rgy_attr_update (
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        unsigned32 num_to_write,
        unsigned32 space_avail,
        sec_attr_t in_attrs[ ],
        unsigned32 *num_returned,
        sec_attr_t out_attrs[ ],
        unsigned32 *num_left,
        signed32 *failure_index,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
A value of type **sec_rgy_domain_t** that identifies the registry domain in which the object specified by *name* resides. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

This parameter is ignored if *name* is **policy** or **replist**.

name A character string of type **sec_rgy_name_t** specifying the name of the person, group, or organization to which the attribute is attached.

num_to_write
A 32-bit unsigned integer that specifies the number of elements in the *in_attrs[ ]* array. This integer must be greater than 0 (zero).

space_avail
Set this parameter to zero. It is a 32-bit unsigned integer that specifies the size of the *out_attrs[ ]* array. Use of this parameter and its associated *out_attrs[ ]* array is reserved for future use by update trigger servers.

in_attrs[ ] An array of values of type **sec_attr_t** that specifies the attribute instances to be updated. The size of the array is determined by *num_to_write*.

Output

num_returned
A pointer to an unsigned 32-bit integer that specifies the number of attribute instances returned in the *out_attrs[ ]* array.

out_attrs[ ] Reserved for future use by update trigger servers.

num_left A pointer to an unsigned 32-bit integer that supplies the number of attributes that could not be returned because of space constraints in the *out_attrs[ ]* buffer. To ensure that all the attributes will be returned, increase the size of the *out_attrs[ ]* array by increasing the size of *space_avail* and *num_returned*.

failure_index
In the event of an error, *failure_index* is a pointer to the element in the *in_attrs[ ]* array that caused the update to fail. If the failure cannot be attributed to a specific attribute, the value of *failure_index* is **−1**.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_attr_update()** routine creates new attribute instances and updates existing attribute instances attached to a object specified by name and registry domain. The instances to be created or updated are passed as an array of **sec_attr_t** data types. This is an atomic operation: if the creation of any attribute in the *in_attrs[ ]* array fails, all updates are aborted. The attribute causing the update to fail is identified in *failure_index*. If the failure cannot be attributed to a given attribute, *failure_index* contains **-1**.

The *in_attrs[ ]* array, which specifies the attributes to be created, contains values of type **sec_attr_t**. These values are as follows:

attr_id A Universal Unique Identifier (UUID) that identifies the attribute type

attr_value Values of **sec_attr_value_t** that specify the attribute's encoding type and values.

If an attribute instance already exists which is identical in both *attr_id* and *attr_value* to an attribute specified in *in_attrs[ ]*, the existing attribute information is overwritten by the new information. For multivalued attributes, every instance with the same *attr_id* is overwritten with the supplied values.

If an attribute instance does not exist, it is created.

For multivalued attributes, because every instance of the multivalued attribute is identified by the same UUID, every instance is overwritten with the supplied value. To change only one of the values, you must supply the values that should be unchanged as well as the new value.

To create instances of multivalued attributes, create individual **sec_attr_t** data types to define each multivalued attribute instance and then pass all of them in in the input array.

Permissions Required

The **sec_rgy_attr_update()** routine requires the **U** (**Update**) permission set for each attribute type identified in the *in_attrs*[ ] array. These permissions are defined as part of the access control list (ACL) manager set in the schema entry of each attribute type.

Files

/usr/include/dce/sec_rgy_attr.idl
The **idl** file from which **dce/sec_rgy_attr.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

unauthorized

database read only

server unavailable

invalid/unsupported attribute type

invalid encoding type

value not unique

attribute instance already exists

trigger server unavailable

site read only

error_status_ok

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_attr_delete(3sec)**, **sec_rgy_attr_test_and_update(3sec)**.

sec_rgy_auth_plcy_get_effective

Purpose

Returns the effective authentication policy for an account

Synopsis

```
#include <dce/policy.h>

void sec_rgy_auth_plcy_get_effective(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *account,
        sec_rgy_plcy_auth_t *auth_policy,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

account A pointer to the account login name (type **sec_rgy_login_name_t**). A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. If all three fields contain empty strings, the authentication policy returned is that of the registry.

Output

auth_policy
A pointer to the **sec_rgy_plcy_auth_t** structure to receive the authentication policy. The authentication policy structure contains the maximum lifetime for an authentication ticket, and the maximum amount of time for which one can be renewed.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_auth_plcy_get_effective()** routine returns the effective authentication policy for the specified account. The authentication policy in effect is the more restrictive of the registry and the account policies for each policy category. If no account is specified, the registry's authentication policy is returned.

Permissions Required

The **sec_rgy_auth_plcy_get_effective()** routine requires the **r** (**read**) permission on the policy object from which the data is to be returned. If an account is specified and an account policy exists, the routine also requires the **r** (**read**) permission on the account principal.

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
The specified account could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_auth_plcy_get_info(3sec)**, **sec_rgy_auth_plcy_set_info(3sec)**.

sec_rgy_auth_plcy_get_info

Purpose

Returns the authentication policy for an account

Synopsis

```
#include <dce/policy.h>

void sec_rgy_auth_plcy_get_info(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *account,
        sec_rgy_plcy_auth_t *auth_policy,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

account A pointer to the account login name (type **sec_rgy_login_name_t**). A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account.

Output

auth_policy
A pointer to the **sec_rgy_plcy_auth_t** structure to receive the authentication policy. The authentication policy structure contains the maximum lifetime for an authentication ticket, and the maximum amount of time for which one can be renewed.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_auth_plcy_get_info()** routine returns the authentication policy for the specified account. If no account is specified, the registry's authentication policy is returned.

Permissions Required

The **sec_rgy_auth_plcy_get_info()** routine requires the **r** (**read**) permission on the policy object or account principal from which the data is to be returned.

Notes

The actual policy in effect will not correspond precisely to what is returned by this call if the overriding registry authentication policy is more restrictive than the policy for the specified account. Use **sec_rgy_auth_plcy_get_effective()** to return the policy currently in effect for the given account.

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No account with the given login name could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_auth_plcy_get_effective(3sec)**, **sec_rgy_auth_plcy_set_info(3sec)**.

sec_rgy_auth_plcy_set_info

Purpose

Sets the authentication policy for an account

Synopsis

```
#include <dce/policy.h>

void sec_rgy_auth_plcy_set_info(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *account,
        sec_rgy_plcy_auth_t *auth_policy,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

account A pointer to the account login name (type **sec_rgy_login_name_t**). A login name is composed of three character strings, containing the principal, group, and organization (PGO) names corresponding to the account. All three names must be completely specified.

auth_policy

A pointer to the **sec_rgy_plcy_auth_t** structure containing the authentication policy. The authentication policy structure contains the maximum lifetime for an authentication ticket, and the maximum amount of time for which one can be renewed.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_auth_plcy_set_info()** routine sets the indicated authentication policy for the specified account. If no account is specified, the authentication policy is set for the registry as a whole.

Permissions Required

The **sec_rgy_auth_plcy_set_info()** routine requires the **a** (*auth_info*) permission on the policy object or account principal for which the data is to be set.

Notes

The policy set on an account may be less restrictive than the policy set for the registry as a whole. In this case, the change in policy has no effect, since the effective policy is the most restrictive combination of the principal and registry authentication policies. (See the **sec_rgy_auth_plcy_get_effective()** routine).

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No account with the given login name could be found.

sec_rgy_not_authorized
The user is not authorized to update the specified record.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_auth_plcy_get_effective(3sec)**, **sec_rgy_auth_plcy_get_info(3sec)**.

sec_rgy_cell_bind

Purpose

Binds to a registry in a cell

Synopsis

```
#include <dce/binding.h>

void sec_rgy_cell_bind(
        unsigned_char_t *cell_name,
        sec_rgy_bind_auth_info_t *auth_info,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

cell_name A character string (type **unsigned_char_t**) containing the name of the cell in question. Upon return, a security server for that cell is associated with *context*, the registry server handle. The cell must be specified completely and precisely. This routine offers none of the pathname resolving services of **sec_rgy_site_bind()**.

auth_info A pointer to the **sec_rgy_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info()** routine).

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (bound to) the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_cell_bind()** routine establishes a relationship with a registry site at an arbitrary level of security. The *cell_name* parameter identifies the target cell.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_bind(3sec)**.

sec_rgy_cursor_reset

Purpose

Resets the registry database cursor

Synopsis

```
#include <dce/misc.h>

void sec_rgy_cursor_reset(
        sec_rgy_cursor_t *cursor);
```

Parameters

Input/Output

cursor A pointer into the registry database.

Description

The **sec_rgy_cursor_reset()** routine resets the database cursor to return the first suitable entry. A cursor is a pointer into the registry. It serves as a place holder when returning successive items from the registry.

A cursor is bound to a particular server. In other words, a cursor that is in use with one replica of the registry has no meaning for any other replica. If a calling program attempts to use a cursor from one replica with another, the cursor is reset and the routine for which the cursor was specified returns the first item in the database.

A cursor that is in use with one call cannot be used with another. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Files

/usr/include/dce/misc.idl
The **idl** file from which **dce/misc.h** was derived.

Examples

The following example illustrates use of the cursor within a loop. The initial **sec_rgy_cursor_reset()** call resets the cursor to point to the first item in the registry. Successive calls to **sec_rgy_pgo_get_next()** return the next PGO item and update the cursor to reflect the last item returned. When the end of the list of PGO items is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter.

```
sec_rgy_cursor_reset(&cursor);
do {
    sec_rgy_pgo_get_next(context, domain, scope, &cursor,
                    &item, name &status);
    if (status == error_status_ok) {
         /* Print formatted PGO item info */
    }
}while (status == error_status_ok);
```

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_get_projlist(3sec)**, **sec_rgy_acct_lookup(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_get_members(3sec)**, **sec_rgy_pgo_get_next(3sec)**.

sec_rgy_login_get_effective

Purpose

Returns the effective login data for an account

Synopsis

```
#include <dce/misc.h>

void sec_rgy_login_get_effective(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_sid_t *sid,
        sec_rgy_unix_sid_t *unix_sid,
        sec_rgy_acct_user_t *user_part,
        sec_rgy_acct_admin_t *admin_part,
        sec_rgy_plcy_t *policy_data,
        signed32 max_number,
        signed32 *supplied_number,
        uuid_t id_projlist[ ],
        signed32 unix_projlist[ ],
        signed32 *num_projects,
        sec_rgy_name_t cell_name,
        uuid_t *cell_uuid,
        sec_override_fields_t *overridden,
        error_status_t *status);
```

Parameters

Input

context The registry server handle.

max_number The maximum number of projects to be returned by the call. This must be no larger than the allocated size of the *projlist[ ]* arrays.

Input/Output

login_name
A pointer to the account login name. A login name is composed of the names for the account's principal, group, and organization (PGO) items.

Output

key_parts
A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

sid A pointer to a **sec_rgy_sid_t** structure to receive the returned subject identifier (SID) for the account. This structure consists of the UUIDs for the account's PGO items.

unix_sid A pointer to a **sec_rgy_unix_sid_t** structure to receive the returned UNIX subject identifier (SID) for the account. This structure consists of the UNIX numbers for the account's PGO items.

user_part A pointer to a **sec_rgy_acct_user_t** structure to receive the returned user data for the account.

admin_part A pointer to a **sec_rgy_acct_admin_t** structure to receive the returned administrative data for the account.

policy_data
A pointer to a **sec_rgy_policy_t** structure to receive the policy data for the account. The policy data is associated with the account's organization, as identified in the login name.

supplied_number
A pointer to the actual number of projects returned. This will always be less than or equal to the *max_number* supplied on input.

id_projlist[ ]
An array to receive the UUID of each project returned. The size allocated for the array is given by *max_number*. If this value is less than the total number of projects in the account project list, multiple calls must be made to return all of the projects.

unix_projlist[ ]
An array to receive the UNIX number of each project returned. The size allocated for the array is given by *max_number*. If this value is less than

the total number of projects in the account project list, multiple calls must be made to return all of the projects.

num_projects
A pointer indicating the total number of projects in the specified account's project list.

cell_name The name of the account's cell.

cell_uuid The UUID for the account's cell.

overridden
A pointer to a 32-bit set of flags identifying the local overrides, if any, for the account login information.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_login_get_effective()** routine returns effective login information for the specified account. Login information is extracted from the account's entry in the registry database. Effective login information is a combination of the login information from the registry database and any login overrides defined for the account on the local machine.

The *overridden* parameter indicates which, if any, of the following local overrides have been defined for the account:

- The UNIX user ID
- The group ID
- The encrypted password
- The account's miscellaneous information (**gecos**) field
- The account's home directory
- The account's login shell

Local overrides for account login information are defined in the **/etc/passwd_override** file and apply only to the local machine.

Files

/usr/include/dce/misc.idl
The **idl** file from which **dce/misc.h** was derived.

/etc/passwd_override
The file that defines local overrides for account login information.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy__object_not_found
The specified account could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_login_get_info(3sec)**.

Files: **passwd_override(5sec)**.

sec_rgy_login_get_info

Purpose

Returns login information for an account

Synopsis

```
#include <dce/misc.h>

void sec_rgy_login_get_info(
        sec_rgy_handle_t context,
        sec_rgy_login_name_t *login_name,
        sec_rgy_acct_key_t *key_parts,
        sec_rgy_sid_t *sid,
        sec_rgy_unix_sid_t *unix_sid,
        sec_rgy_acct_user_t *user_part,
        sec_rgy_acct_admin_t *admin_part,
        sec_rgy_plcy_t *policy_data,
        signed32 max_number,
        signed32 *supplied_number,
        uuid_t id_projlist[ ],
        signed32 unix_projlist[ ],
        signed32 *num_projects,
        sec_rgy_name_t cell_name,
        uuid_t *cell_uuid,
        error_status_t *status);
```

Parameters

Input

context The registry server handle.

max_number
The maximum number of projects to be returned by the call. This must be no larger than the allocated size of the *projlist[ ]* arrays.

Input/Output

login_name
A pointer to the account login name. A login name is composed of the names for the account's principal, group, and organization (PGO) items.

Output

key_parts A pointer to the minimum abbreviation allowed when logging in to the account. Abbreviations are not currently implemented and the only legal value is **sec_rgy_acct_key_person**.

sid A pointer to a **sec_rgy_sid_t** structure to receive the UUID's representing the account's PGO items.

unix_sid A pointer to a **sec_rgy_unix_sid_t** structure to receive the UNIX numbers for the account's PGO items.

user_part A pointer to a **sec_rgy_acct_user_t** structure to receive the returned user data for the account.

admin_part A pointer to a **sec_rgy_acct_admin_t** structure to receive the returned administrative data for the account.

policy_data
A pointer to a **sec_rgy_policy_t** structure to receive the policy data for the account. The policy data is associated with the account's organization, as identified in the login name.

supplied_number
A pointer to the actual number of projects returned. This will always be less than or equal to the *max_number* supplied on input.

id_projlist[ ]
An array to receive the UUID of each project returned. The size allocated for the array is given by *max_number*. If this value is less than the total number of projects in the account project list, multiple calls must be made to return all of the projects.

unix_projlist[ ]
An array to receive the UNIX number of each project returned. The size allocated for the array is given by *max_number*. If this value is less than the total number of projects in the account project list, multiple calls must be made to return all of the projects.

num_projects
: A pointer indicating the total number of projects in the specified account's project list.

cell_name
: The name of the account's cell.

cell_uuid
: The UUID for the account's cell.

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_login_get_info()** routine returns login information for the specified account. This information is extracted from the account's entry in the registry database. To return any local overrides for the account's login data, use **sec_rgy_login_get_effective()**.

Permissions Required

The **sec_rgy_login_get_info()** routine requires the **r** (**read**) permission on the account principal from which the data is to be returned.

Files

/usr/lib/dce/misc.idl
: The **idl** file from which **dce/misc.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
: The specified account could not be found.

sec_rgy_server_unavailable
: The DCE registry server is unavailable.

error_status_ok
: The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_login_get_effective(3sec)**.

sec_rgy_pgo_add

Purpose

Adds a PGO item to the registry database

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_add(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        sec_rgy_pgo_item_t *pgo_item,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

name A pointer to a **sec_rgy_name_t** character string containing the name of the new PGO item.

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure containing the data for the new PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item may have (or belong to) a concurrent group set.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_add()** routine adds a PGO item to the registry database.

The PGO data consists of the following:

- The universal unique identifier (UUID) of the PGO item. Specify NULL to have the registry server create a new UUID for an item.
- The UNIX number for the PGO item. Since the registry uses embedded UNIX IDs (where a subset of the UUID bits represent the UNIX ID), the specified ID must match the UUID, if both are specified.
- The quota for subaccounts allowed for this item entry.
- The full name of the PGO item.
- Flags (in the **sec_rgy_pgo_flags_t** format) indicating whether
  - A principal item is an alias.
  - The PGO item can be deleted from the registry.
  - A principal item can have a concurrent group set.
  - A group item can appear in a concurrent group set.

Permissions Required

The **sec_rgy_pgo_add()** routine requires the **i** (**insert**) permission on the parent directory in which the the PGO item is to be created.

Notes

An account can be added to the registry database only when all its constituent PGO items are already in the database, and the appropriate membership relationships between them are established. For example, to establish an account with principal name **tom**, group name **writers**, and organization name **hp**, all three names must exist as independent PGO items in the database. Furthermore, **tom** must be a member of **writers**, which must be a member of **hp**. (See **sec_rgy_acct_add()** to add an account to the registry.)

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to add the specified PGO item.

sec_rgy_object_exists
A PGO item already exists with the name given in *name*.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_add(3sec)**, **sec_rgy_pgo_delete(3sec)**, **sec_rgy_pgo_rename(3sec)**, **sec_rgy_pgo_replace(3sec)**.

sec_rgy_pgo_add_member

Purpose

Adds a principal to a group or organization

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_add_member(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t go_name,
        sec_rgy_name_t principal_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_group
The *go_name* parameter identifies a group.

sec_rgy_domain_org
The *go_name* parameter identifies an organization.

go_name A character string (type **sec_rgy_name_t**) containing the name of the group or organization to which the specified principal will be added.

principal_name
A character string (type **sec_rgy_name_t**) containing the name of the principal to be added to the membership list of the group or organization specified by *go_name*. You must use fully qualified names to add foreign

principals as members of a group. (Only local principals can be added as members of an organization.)

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_add_member()** routine adds a member to the membership list of a group or organization in the registry database. For this call to succeed when adding a principal from a foreign cell to a group, the Security Server (**secd**) must be running in the foreign cell.

Permissions Required

The **sec_rgy_pgo_add_member()** routine requires the **M** (**Member_list**) permission on the group or organization item specified by *go_name*. If *go_name* specifies a group, the routine also requires the **g** (**groups**) permission on the principal identified by *principal_name*.

Notes

An account can be added to the registry database only when all its constituent PGO items are already in the database, and the appropriate membership relationships between them are established. For example, to establish an account with principal name **tom**, group name **writers**, and organization name **hp**, all three names must exist as independent PGO items in the database. Furthermore, **tom** must be a member of **writers**, which must be a member of **hp**. (See the **sec_rgy_acct_add()** routine to add an account to the registry.)

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
: The client program is not authorized to add members to the specified group or organization.

sec_rgy_bad_domain
: An invalid domain was specified. A member can be added only to a group or organization, not a principal.

sec_rgy_object_not_found
: The registry server could not find the specified name.

sec_rgy_server_unavailable
: The DCE registry server is unavailable.

error_status_ok
: The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_delete_member(3sec)**, **sec_rgy_pgo_get_members(3sec)**, **sec_rgy_pgo_is_member(3sec)**.

sec_rgy_pgo_delete

Purpose

Deletes a PGO item from the registry database

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_delete(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

name A pointer to a **sec_rgy_name_t** character string containing the name of the PGO item to be deleted.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_delete()** routine deletes a PGO item from the registry database. Any account depending on the deleted PGO item is also deleted.

Permissions Required

The **sec_rgy_pgo_delete()** routine requires the following permissions:

- The **d** (**delete**) permission on the parent directory that contains the the PGO item to be deleted.
- The **D** (**Delete_object**) permission on the PGO item itself.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to delete the specified item.

sec_rgy_object_not_found
The registry server could not find the specified item.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**.

sec_rgy_pgo_delete_member

Purpose

Removes a member from a group or organization

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_delete_member(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t go_name,
        sec_rgy_name_t principal_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_group
The *go_name* parameter identifies a group.

sec_rgy_domain_org
The *go_name* parameter identifies an organization.

go_name A character string (type **sec_rgy_name_t**) containing the name of the group or organization from which the specified principal will be removed.

principal_name
A character string (type **sec_rgy_name_t**) containing the name of the principal to be removed from the membership list of the group or

organization specified by *go_name*. You must use fully-qualified names to remove foreign principals from groups. (Only local principals can be members of an organization.)

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_delete_member()** routine removes a member from the membership list of a group or organization. The principal to be removed from a group can be in the local or a foreign cell. (Only local principals can be members of an organization.)

Permissions Required

The **sec_rgy_pgo_delete_member()** routine requires the **M** (**Member_list**) permission on the group or organization item specified by *go_name*.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to remove the specified member.

sec_rgy_bad_domain
An invalid domain was specified. Members can exist only for groups and organizations, not for principals.

sec_rgy_object_not_found
The specified group or organization was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_add_member**.

sec_rgy_pgo_get_by_eff_unix_num

Purpose

Returns the name and data for a PGO item identified by its effective UNIX number

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_by_eff_unix_num(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t scope,
        signed32 unix_id,
        boolean32 allow_aliases,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_pgo_item_t *pgo_item,
        sec_rgy_name_t name,
        boolean32 *overridden,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The UNIX number identifies a principal.

sec_rgy_domain_group
The UNIX number identifies a group.

Note that this function does *not* support the value **sec_rgy_domain_org**.

scope A character string (type **sec_rgy_name_t**) containing the scope of the desired search. The registry database is designed to accommodate a tree-structured name hierarchy. The scope of a search is the name of the branch under which the search takes place. For example, all names in a registry might start with **/alpha**, and be divided further into **/beta** or **/gamma**. To search only the part of the database under **/beta**, the scope of the search would be **/alpha/beta**, and any resulting PGO items would have names beginning with this string. Note that these naming conventions need not have anything to do with group or organization PGO item membership lists.

unix_id The UNIX number of the desired registry PGO item.

allow_aliases

A **boolean32** value indicating whether to search for a primary PGO item, or whether the search can be satisfied with an alias. If TRUE, the routine returns the next entry found for the PGO item. If FALSE, the routine returns only the primary entry.

Input/Output

item_cursor

An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_pgo_get_next()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

pgo_item

A pointer to a **sec_rgy_pgo_item_t** structure to receive the data for the returned PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set. The data is as it appears in the registry, for that UNIX number, even though some of the fields may have been overridden locally.

name A pointer to a **sec_rgy_name_t** character string containing the returned name for the PGO item. This string might contain a local override value if the supplied UNIX number is found in the **passwd_override** or **group_override** file.

overridden
: A pointer to a **boolean32** value indicating whether or not the supplied UNIX number has an entry in the local override file (**passwd_override** or **group_override**).

status
: A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_by_eff_unix_num()** routine returns the name and data for a PGO item. The desired item is identified by its type (domain) and its UNIX number.

This routine is similar to the **sec_rgy_pgo_get_by_unix_num()** routine. The difference between the routines is that **sec_rgy_pgo_get_by_eff_unix_num()** first searches the local override files for the respective *name_domain* for a match with the supplied UNIX number. If an override match is found, and an account or group name is found in that entry, then that name is used to obtain PGO data from the registry and the value of the *overridden* parameter is set to TRUE.

The *item_cursor* parameter specifies the starting point for the search through the registry database. It provides an automatic place holder in the database. The routine automatically updates this variable to point to the next PGO item after the returned item. The returned cursor location can be supplied on a subsequent database access call that also uses a PGO item cursor.

Permissions Required

The **sec_rgy_pgo_get_by_eff_unix_num()** routine requires the **r** (**read**) permission on the PGO item to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

group_override
The local group override file.

passwd_override
The local password override file.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of PGO items.

sec_rgy_object_not_found
The specified PGO item was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_get_next(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_get_by_id

Purpose

Returns the name and data for a PGO item identified by its UUID

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_by_id(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t scope,
        uuid_t *item_id,
        boolean32 allow_aliases,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_pgo_item_t *pgo_item,
        sec_rgy_name_t name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The UUID identifies a principal.

sec_rgy_domain_group
The UUID identifies a group.

sec_rgy_domain_org
The UUID identifies an organization.

scope A character string (type **sec_rgy_name_t**) containing the scope of the desired search. The registry database is designed to accommodate a tree-structured name hierarchy. The scope of a search is the name of the branch under which the search takes place. For example, all names in a registry might start with **/alpha**, and be divided further into **/beta** or **/gamma**. To search only the part of the database under **/beta**, the scope of the search would be **/alpha/beta**, and any resulting PGO items would have names beginning with this string. Note that these naming conventions need not have anything to do with group or organization PGO item membership lists.

item_id A pointer to the **uuid_t** variable containing the UUID (Unique Universal Identifier) of the desired PGO item.

allow_aliases
A **boolean32** value indicating whether to search for a primary PGO item, or whether the search can be satisfied with an alias. If TRUE, the routine returns the next entry found for the PGO item. If FALSE, the routine returns only the primary entry.

Input/Output

item_cursor
An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_pgo_get_by_id()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure to receive the data for the returned PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set.

name A pointer to a **sec_rgy_name_t** character string containing the returned name for the PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_by_id()** routine returns the name and data for a PGO item. The desired item is identified by its type (domain) and its UUID.

The *item_cursor* parameter specifies the starting point for the search through the registry database. It provides an automatic place holder in the database. The routine automatically updates this variable to point to the next PGO item after the returned item. The returned cursor location can be supplied on a subsequent database access call that also uses a PGO item cursor.

Permissions Required

The **sec_rgy_pgo_get_by_id()** routine requires the **r** (**read**) permission on the PGO item to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of PGO items.

sec_rgy_object_not_found
The specified PGO item was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_get_next(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_get_by_name

Purpose

Returns the data for a named PGO item

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_by_name(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t pgo_name,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_pgo_item_t *pgo_item,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

pgo_name A character string (type **sec_rgy_name_t**) containing the name of the principal, group, or organization to search for.

Input/Output

item_cursor

An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_pgo_get_by_name()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure to receive the data for the returned PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_by_name()** routine returns the data for a named PGO item from the registry database. The desired item is identified by its type (*name_domain*) and name.

The *item_cursor* parameter specifies the starting point for the search through the registry database. It provides an automatic place holder in the database. The routine automatically updates this variable to point to the next PGO item after the returned item. The returned cursor location can be supplied on a subsequent database access call that also uses a PGO item cursor.

Permissions Required

The **sec_rgy_pgo_get_by_name()** routine requires the **r** (**read**) permission on the PGO item to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort

of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of PGO items.

sec_rgy_object_not_found
The specified PGO item was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_get_next(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_get_by_unix_num

Purpose

Returns the name and data for a PGO item identified by its UNIX ID

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_by_unix_num(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t scope,
        signed32 unix_id,
        boolean32 allow_aliases,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_pgo_item_t *pgo_item,
        sec_rgy_name_t name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The UNIX number identifies a principal.

sec_rgy_domain_group
The UNIX number identifies a group.

sec_rgy_domain_org
The UNIX number identifies an organization.

scope A character string (type **sec_rgy_name_t**) containing the scope of the desired search. The registry database is designed to accommodate a tree-structured name hierarchy. The scope of a search is the name of the branch under which the search takes place. For example, all names in a registry might start with **/alpha**, and be divided further into **/beta** or **/gamma**. To search only the part of the database under **/beta**, the scope of the search would be **/alpha/beta**, and any resulting PGO items would have names beginning with this string. Note that these naming conventions need not have anything to do with group or organization PGO item membership lists.

unix_id The UNIX number of the desired registry PGO item.

allow_aliases
A **boolean32** value indicating whether to search for a primary PGO item, or whether the search can be satisfied with an alias. If TRUE, the routine returns the next entry found for the PGO item. If FALSE, the routine returns only the primary entry.

Input/Output

item_cursor
An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_pgo_get_by_unix_num()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure to receive the data for the returned PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set.

name A pointer to a **sec_rgy_name_t** character string containing the returned name for the PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_by_unix_num()** routine returns the name and data for a PGO item. The desired item is identified by its type (domain) and its UNIX number.

The *item_cursor* parameter specifies the starting point for the search through the registry database. It provides an automatic place holder in the database. The routine automatically updates this variable to point to the next PGO item after the returned item. The returned cursor location can be supplied on a subsequent database access call that also uses a PGO item cursor.

Permissions Required

The **sec_rgy_pgo_get_by_unix_num()** routine requires the **r** (**read**) permission on the PGO item to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of PGO items.

sec_rgy_object_not_found
The specified PGO item was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_next(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_get_members

Purpose

Returns the membership list for a specified group or organization or returns the set of groups in which the specified principal is a member

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_members(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t go_name,
        sec_rgy_cursor_t *member_cursor,
        signed32 max_members,
        sec_rgy_member_t member_list[ ],
        signed32 *number_supplied,
        signed32 *number_members,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a **secd** server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable specifies whether *go_name* identifies a principal, group, or organization. The valid values are as follows:

sec_rgy_domain_group

The *go_name* parameter identifies a group.

sec_rgy_domain_org

The *go_name* parameter identifies an organization.

sec_rgy_domain_person
The *go_name* parameter identifies an principal.

go_name A character string (type **sec_rgy_name_t**) that contains the name of a group, organization, or principal. If *go_name* is the name of a group or organization, the call returns the group's or organization's member list. If *go_name* is the name of a principal, the call returns a list of all groups in which the principal is a member. (Contrast this with the **sec_rgy_acct_get_proj** call, which returns only those groups in which the principal is a member and that have been marked to be included in the principal's project list.)

max_members
A **signed32** variable containing the allocated dimension of the *member_list[ ]* array. This is the maximum number of members or groups that can be returned by a single call.

Input/Output

member_cursor
An opaque pointer to a specific entry in the membership list or list of groups. The returned list begins with the entry specified by *member_cursor*. Upon return, the cursor points to the next entry after the last one returned. If there are no more entries, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor to the beginning of the list.

Output

member_list[ ]
An array of character strings to receive the returned member or group names. The size allocated for the array is given by *max_number*. If this value is less than the total number of members or group names, multiple calls must be made to return all of the members or groups. For groups, fully qualified names are returned for foreign principals and local names for local principals. (Only local principals can be members of an organization.)

number_supplied
A pointer to a **signed32** variable to receive the number of members or groups actually returned in *member_list[ ]*.

number_members
: A pointer to a **signed32** variable to receive the total number of members or groups. If this number is greater than *number_supplied*, multiple calls to **sec_rgy_pgo_get_members()** are necessary. Use the *member_cursor* parameter to coordinate successive calls.

status
: A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_members()** routine returns a list of the members in the specified group or organization, or a list of groups in which a specified principal is a member.

The *member_cursor* parameter specifies the starting point for the search through the registry database. It provides an automatic place holder in the database. The routine automatically updates *member_cursor* to point to the next member or group (if any) after the returned list. If not all of the members or groups are returned, the updated cursor can be supplied on successive calls to return the remainder of the list.

Permissions Required

The **sec_rgy_pgo_get_members()** routine requires the **r** (**read**) permission on the group, organization, or principal object specified by *go_name*.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Return Values

The routine returns

- The names of the groups or members in *member_list[ ]*
- The number of members or groups returned by the call in *number_supplied*
- The total number of members in the group or organization, or the total number of groups in which the principal is a member in *number_members*

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor points to the end of the membership list for a group or organization or to the end of the list of groups for a principal.

sec_rgy_object_not_found
The specified group, organization, or principal could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_acct_get_proj(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add_member(3sec)**, **sec_rgy_pgo_is_member(3sec)**.

sec_rgy_pgo_get_next

Purpose

Returns the next PGO item in the registry database

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_get_next(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t scope,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_pgo_item_t *pgo_item,
        sec_rgy_name_t name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
Returns the next principal item.

sec_rgy_domain_group
Returns the next group item.

sec_rgy_domain_org
Returns the next organization item.

scope A character string (type **sec_rgy_name_t**) containing the scope of the desired search. The registry database is designed to accommodate a

tree-structured name hierarchy. The scope of a search is the name of the branch under which the search takes place. For example, all names in a registry might start with **/alpha**, and be divided further into **/beta** or **/gamma**. To search only the part of the database under **/beta**, the scope of the search would be **/alpha/beta**, and any resulting PGO items would have names beginning with this string. Note that these naming conventions need not have anything to do with group or organization PGO item membership lists.

Input/Output

item_cursor

An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_pgo_get_next()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns the value **sec_rgy_no_more_entries** in the *status* parameter. Use **sec_rgy_cursor_reset()** to reset the cursor.

Output

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure to receive the data for the returned PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set.

name A pointer to a **sec_rgy_name_t** character string containing the name of the returned PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_get_next()** routine returns the data and name for the PGO in the registry database indicated by *item_cursor*. It also advances the cursor to point to the next PGO item in the database. Successive calls to this routine return all the PGO items in the database of the specified type (given by *name_domain*), in storage order.

The PGO data consists of the following:

- The universal unique identifier (UUID) of the PGO item.
- The UNIX number for the PGO item.

- The quota for subaccounts.
- The full name of the PGO item.
- Flags indicating whether
  - A principal item is an alias.
  - The PGO item can be deleted.
  - A principal item can have a concurrent group set.
  - A group item can appear on a concurrent group set.

Permissions Required

The **sec_rgy_pgo_get_next()** routine requires the **r** (**read**) permission on the PGO item to be viewed.

Cautions

There are several different types of cursors used in the registry application programmer interface (API). Some cursors point to PGO items, others point to members in a membership list, and others point to account data. Do not use a cursor for one sort of object in a call expecting another sort of object. For example, you cannot use the same cursor on a call to **sec_rgy_acct_get_projlist()** and **sec_rgy_pgo_get_next()**. The behavior in this case is undefined.

Furthermore, cursors are specific to a server. A cursor pointing into one replica of the registry database is useless as a pointer into another replica.

Use **sec_rgy_cursor_reset()** to renew a cursor for use with another call or for another server.

Return Values

The routine returns the data for the returned PGO item in *pgo_item* and the name in *name*.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The cursor is at the end of the list of PGO items.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cursor_reset(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_id_to_name

Purpose

Returns the name for a PGO item identified by its UUID

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_id_to_name(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        uuid_t *item_id,
        sec_rgy_name_t pgo_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The *item_id* parameter identifies a principal.

sec_rgy_domain_group
The *item_id* parameter identifies a group.

sec_rgy_domain_org
The *item_id* parameter identifies an organization.

item_id A pointer to the **uuid_t** variable containing the input UUID (unique universal identifier).

Output

pgo_name A character string (type **sec_rgy_name_t**) containing the name of the principal, group, or organization with the input UUID.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_id_to_name()** routine returns the name of the PGO item having the specified UUID.

Permissions Required

The **sec_rgy_pgo_id_to_name()** routine requires at least one permission of any kind on the PGO item to be viewed.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No item with the specified UUID could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_id_to_unix_num

Purpose

Returns the UNIX number for a PGO item identified by its UUID

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_id_to_unix_num(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        uuid_t *item_id,
        signed32 *item_unix_id,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The *item_id* parameter identifies a principal.

sec_rgy_domain_group
The *item_id* parameter identifies a group.

sec_rgy_domain_org
The *item_id* parameter identifies an organization.

item_id A pointer to the **uuid_t** variable containing the input UUID (unique universal identifier).

Output

item_unix_id
A pointer to the **signed32** variable to receive the returned UNIX number for the PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_id_to_unix_num()** routine returns the UNIX number for the PGO item having the specified UUID.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No item with the specified UUID could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_is_member

Purpose

Checks group or organization membership

Synopsis

```
#include <dce/pgo.h>

boolean32 sec_rgy_pgo_is_member(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t go_name,
        sec_rgy_name_t principal_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_group
The *go_name* parameter identifies a group.

sec_rgy_domain_org
The *go_name* parameter identifies an organization.

go_name A character string (type **sec_rgy_name_t**) containing the name of the group or organization whose membership list is in question.

principal_name
A character string (type **sec_rgy_name_t**) containing the name of the principal whose membership in the group or organization specified by

go_name is in question. For groups, use fully-qualified names for foreign principals. (Only local principals can be members of an organization.)

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_is_member()** routine tests whether the specified principal is a member of the named group or organization.

Permissions Required

The **sec_rgy_pgo_is_member()** routine requires the **t** (**test**) permission on the group or organization item specified by *go_name*.

Return Values

The routine returns TRUE if the principal is a member of the named group or organization. If the principal is not a member, the routine returns FALSE.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
The named group or organization was not found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add_member(3sec)**, **sec_rgy_pgo_get_members(3sec)**.

sec_rgy_pgo_name_to_id

Purpose

Returns the UUID for a named PGO item

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_name_to_id(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t pgo_name,
        uuid_t *item_id,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

pgo_name A character string (type **sec_rgy_name_t**) containing the name of the principal, group, or organization whose UUID is desired.

Output

item_id A pointer to the **uuid_t** variable containing the UUID (unique universal identifier) of the resulting PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_name_to_id()** routine returns the UUID associated with the named PGO item.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
The specified PGO item could not be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_name_to_unix_num

Purpose

Returns the UNIX number for a PGO item identified by its name

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_name_to_unix_num(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t pgo_name,
        signed32 *item_unix_id,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

pgo_name A character string containing the name of the PGO item in question.

Output

item_unix_id
A pointer to the **signed32** variable to receive the returned UNIX number for the PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_name_to_unix_num()** routine returns the UNIX number for the PGO item having the specified name.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No item with the specified UUID could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**, **sec_rgy_pgo_unix_num_to_id(3sec)**.

sec_rgy_pgo_rename

Purpose

Changes the name of a PGO item in the registry database

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_rename(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t old_name,
        sec_rgy_name_t new_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

old_name A pointer to a **sec_rgy_name_t** character string containing the existing name of the PGO item.

new_name A pointer to a **sec_rgy_name_t** character string containing the new name for the PGO item.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_rename()** routine renames a PGO item in the registry database.

Permissions Required

If the **sec_rgy_pgo_rename()** routine is performing a rename within a directory, it requires the **n** (**name**) permission on the old name of the PGO item. If the routine is performing a move between directories, it requires the following permissions:

- The **d** (**delete**) permission on the parent directory that contains the PGO item.
- The **n** (**name**) permission on the old name of the PGO item.
- The **i** (**insert**) permission on the parent directory in which the PGO item is to be added under the new name.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to change the name of the specified PGO item.

sec_rgy_object_not_found
The registry server could not find the specified PGO item.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_replace(3sec)**.

sec_rgy_pgo_replace

Purpose

Replaces the data in an existing PGO item

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_replace(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        sec_rgy_name_t pgo_name,
        sec_rgy_pgo_item_t *pgo_item,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain

This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The name identifies a principal.

sec_rgy_domain_group
The name identifies a group.

sec_rgy_domain_org
The name identifies an organization.

pgo_name A character string (type **sec_rgy_name_t**) containing the name of the principal, group, or organization whose data is to be replaced.

pgo_item A pointer to a **sec_rgy_pgo_item_t** structure containing the new data for the PGO item. The data in this structure includes the PGO item's name, UUID, UNIX number (if any), and administrative data, such as whether the item, if a principal, may have a concurrent group set.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_replace()** routine replaces the data associated with a PGO item in the registry database.

The UNIX ID and UUID of a PGO item cannot be replaced. To change the UNIX ID or UUID, the existing PGO item must be deleted and a new PGO item added in its place. The one exception to this rule is that the UNIX ID can be replaced in the PGO item for a cell principal. The reason for this exception is that the UUID for a cell principal does not contain an embedded UNIX ID.

Permissions Required

The **sec_rgy_pgo_replace()** routine requires at least one of the following permissions:

- The **m** (**mgmt_info**) permission on the PGO item, if **quota** or **flags** is being set.
- The **f** (**fullname**) permission on the PGO item, if **fullname** is being set.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The client program is not authorized to replace the specified PGO item.

sec_rgy_object_not_found
No PGO item was found with the given name.

sec_rgy_unix_id_changed
The UNIX number of the PGO item was changed.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_delete(3sec)**, **sec_rgy_pgo_rename(3sec)**.

sec_rgy_pgo_unix_num_to_id

Purpose

Returns the UUID for a PGO item identified by its UNIX number

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_unix_num_to_id(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        signed32 item_unix_id,
        uuid_t *item_id,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
This variable identifies the type of the principal, group, or organization (PGO) item identified by the given name. The valid values are as follows:

sec_rgy_domain_person
The *item_unix_id* parameter identifies a principal.

sec_rgy_domain_group
The *item_unix_id* parameter identifies a group.

sec_rgy_domain_org
The *item_unix_id* parameter identifies an organization.

item_unix_id
The **signed32** variable containing the UNIX number for the PGO item.

Output

item_id A pointer to the **uuid_t** variable containing the UUID (unique universal identifier) of the resulting PGO item.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_unix_num_to_id()** routine returns the universal unique identifier (UUID) for a PGO item that has the specified UNIX number.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No item with the specified UNIX number could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**, **sec_rgy_pgo_id_to_name(3sec)**, **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**.

sec_rgy_pgo_unix_num_to_name

Purpose

Returns the name for a PGO item identified by its UNIX number

Synopsis

```
#include <dce/pgo.h>

void sec_rgy_pgo_unix_num_to_name(
        sec_rgy_handle_t context,
        sec_rgy_domain_t name_domain,
        signed32 item_unix_id,
        sec_rgy_name_t pgo_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name_domain
The type of the principal, group, or organization (PGO) item identified by *item_unix_id*. Valid values are as follows:

sec_rgy_domain_person
The *item_unix_id* parameter identifies a principal.

sec_rgy_domain_group
The *item_unix_id* parameter identifies a group.

sec_rgy_domain_org
The *item_unix_id* parameter identifies an organization.

item_unix_id
The **signed32** variable containing the UNIX number for the PGO item.

Output

pgo_name A character string containing the name of the PGO item in question.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_pgo_unix_num_to_name()** routine returns the name for a PGO item that has the specified UNIX number.

Permissions Required

The **sec_rgy_pgo_unix_num_to_name()** routine requires at least one permission of any kind on the PGO item identified by *item_unix_id*.

Files

/usr/include/dce/pgo.idl
The **idl** file from which **dce/pgo.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
No item with the specified UNIX number could be found.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_pgo_add(3sec)**, **sec_rgy_pgo_get_by_id(3sec)**, **sec_rgy_pgo_get_by_name(3sec)**, **sec_rgy_pgo_get_by_unix_num(3sec)**,

sec_rgy_pgo_id_to_name(3sec), **sec_rgy_pgo_id_to_unix_num(3sec)**, **sec_rgy_pgo_name_to_id(3sec)**.

sec_rgy_plcy_get_effective

Purpose

Returns the effective policy for an organization

Synopsis

```
#include <dce/policy.h>

void sec_rgy_plcy_get_effective(
        sec_rgy_handle_t context,
        sec_rgy_name_t organization,
        sec_rgy_plcy_t *policy_data,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

organization
A character string (type **sec_rgy_name_t**) containing the name of the organization for which the policy data is to be returned. If this string is empty, the routine returns the registry's policy data.

Output

policy_data
A pointer to the **sec_rgy_plcy_t** structure to receive the authentication policy. This structure contains the minimum length of a user's password, the lifetime of a password, the expiration date of a password, the lifetime of the entire account, and some flags describing limitations on the password spelling.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_plcy_get_effective()** routine returns the effective policy for the specified organization.

The effective policy data is the most restrictive combination of the registry and the organization policies.

The policy data consists of the following:

- The password expiration date. This is the date on which account passwords will expire.
- The minimum length allowed for account passwords.
- The period of time (life span) for which account passwords will be valid.
- The period of time (life span) for which accounts will be valid.
- Flags indicating whether account passwords can consist entirely of spaces or entirely of alphanumeric characters.

Permissions Required

The **sec_rgy_plcy_get_effective()** routine requires the **r** (**read**) permission on the policy object from which the data is to be returned. If an organization is specified, the routine also requires the **r** (**read**) permission on the organization.

Notes

If no organization is specified, the routine returns the registry's policy data. To return the effective policy, an organization must be specified. This is because the routine compares the registry's policy data with that of the organization to determine which is more restrictive.

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
The registry server could not find the specified organization.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_plcy_get_info(3sec)**, **sec_rgy_plcy_set_info(3sec)**.

sec_rgy_plcy_get_info

Purpose

Returns the policy for an organization

Synopsis

```
#include <dce/policy.h>

void sec_rgy_plcy_get_info(
        sec_rgy_handle_t context,
        sec_rgy_name_t organization,
        sec_rgy_plcy_t *policy_data,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

organization

A character string (type **sec_rgy_name_t**) containing the name of the organization for which the policy data is to be returned. If this string is empty, the routine returns the registry's policy data.

Output

policy_data

A pointer to the **sec_rgy_plcy_t** structure to receive the authentication policy. This structure contains the minimum length of a user's password, the lifetime of a password, the expiration date of a password, the lifetime of the entire account, and some flags describing limitations on the password spelling.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_plcy_get_info()** routine returns the policy data for the specified organization. If no organization is specified, the registry's policy data is returned.

The policy data consists of the following:

- The password expiration date. This is the date on which account passwords will expire.
- The minimum length allowed for account passwords.
- The period of time (life span) for which account passwords will be valid.
- The period of time (life span) for which accounts will be valid.
- Flags indicating whether account passwords can consist entirely of spaces or entirely of alphanumeric characters.

Permissions Required

The **sec_rgy_plcy_get_info()** routine requires the **r** (**read**) permission on the policy object or organization from which the data is to be returned.

Notes

The returned policy may not be in effect if the overriding registry authorization policy is more restrictive. (See the **sec_rgy_auth_plcy_get_effective()** routine.)

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_object_not_found
The registry server could not find the specified organization or the organization exists, but policy has not been set for the the specified organizaton.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_plcy_get_effective_info(3sec)**, **sec_rgy_plcy_set_info(3sec)**.

sec_rgy_plcy_set_info

Purpose

Sets the policy for an organization

Synopsis

```
#include <dce/policy.h>

void sec_rgy_plcy_set_info(
        sec_rgy_handle_t context,
        sec_rgy_name_t organization,
        sec_rgy_plcy_t *policy_data,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

organization

A character string (type **sec_rgy_name_t**) containing the name of the organization for which the policy data is to be returned. If this string is empty, the routine sets the registry's policy data.

policy_data

A pointer to the **sec_rgy_plcy_t** structure containing the authentication policy. This structure contains the minimum length of a user's password, the lifetime of a password, the expiration date of a password, the lifetime of the entire account, and some flags describing limitations on the password spelling.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_plcy_set_info()** routine sets the authentication policy for a specified organization. If no organization is specified, the registry's policy data is set.

Policy data can be returned or set for individual organizations and for the registry as a whole.

Permissions Required

The **sec_rgy_plcy_set_info()** routine requires the **m** (**mgmt_info**) permission on the policy object or organization for which the data is to be set.

Notes

The policy set on an account may be less restrictive than the policy set for the registry as a whole. In this case, the changes in policy have no effect, since the effective policy is the most restrictive combination of the organization and registry authentication policies. (See the **sec_rgy_auth_plcy_get_effective()** routine.)

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The user is not authorized to perform this operation.

sec_rgy_object_not_found
The registry server could not find the specified organization.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_plcy_get_effective(3sec)**, **sec_rgy_plcy_get_info(3sec)**.

sec_rgy_properties_get_info

Purpose

Returns registry properties

Synopsis

```
#include <dce/policy.h>

void sec_rgy_properties_get_info(
        sec_rgy_handle_t context,
        sec_rgy_properties_t *properties,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

Output

properties

A pointer to a **sec_rgy_properties_t** structure to receive the returned property information. A registry's property information contains information such as the default and minimum lifetime and other restrictions on privilege attribute certificates, the realm authentication name, and whether or not this replica of the registry supports updates.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_properties_get_info()** routine returns a list of the registry properties.

The property information consists of the following:

read_version
A stamp specifying the earliest version of the registry server software that can read from this registry.

write_version
A stamp specifying the earliest version of the registry server software that can write to this registry.

minimum_ticket_lifetime
The minimum period of time for which an authentication ticket remains valid.

default_certificate_lifetime
The default period of time for which an authentication certificate (ticket-granting ticket) remains valid. A process can request an authentication certificate with a longer lifetime. Note that the maximum lifetime for an authentication certificate cannot exceed the lifetime established by the effective policy for the requesting account.

low_unix_id_person
The lowest UNIX ID that can be assigned to a principal in the registry.

low_unix_id_group
The lowest UNIX ID that can be assigned to a group in the registry.

low_unix_id_org
The lowest UNIX ID that can be assigned to an organization in the registry.

max_unix_id
The maximum UNIX ID that can be used for any item in the registry.

realm A character string naming the cell controlled by this registry.

realm_uuid The UUID of the cell controlled by this registry.

flags Flags include the following:

sec_rgy_prop_readonly
If TRUE, the registry database is read-only.

sec_rgy_prop_auth_cert_unbound
If TRUE, privilege attribute certificates can be generated for use at any site.

sec_rgy_prop_shadow_password
If FALSE, passwords can be distributed over the network. If this flag is TRUE, passwords will be stripped from the returned data to the **sec_rgy_acct_lookup()**, and other calls that return an account's encoded password.

sec_rgy_prop_embedded_unix_id
All registry UUIDs contain embedded UNIX IDs. This implies that the UNIX ID of any registry object cannot be changed, since UUIDs are immutable.

Permissions Required

The **sec_rgy_properties_get_info()** routine requires the **r** (**read**) permission on the policy object from which the property information is to be returned.

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_properties_set_info(3sec)**.

sec_rgy_properties_set_info

Purpose

Sets registry properties

Synopsis

```
#include <dce/policy.h>

void sec_rgy_properties_set_info(
        sec_rgy_handle_t context,
        sec_rgy_properties_t *properties,
        error_status_t *status);
```

Parameters

Input

context The registry server handle. An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

properties A pointer to a **sec_rgy_properties_t** structure containing the registry property information to be set. A registry's property information contains information such as the default and minimum lifetime and other restrictions on privilege attribute certificates, the realm authentication name, and whether or not this replica of the registry supports updates.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_properties_set_info()** routine sets the registry properties.

The property information consists of the following:

read_version
A stamp specifying the earliest version of the registry server software that can read from this registry.

write_version
A stamp specifying the earliest version of the registry server software that can write to this registry.

minimum_ticket_lifetime
The minimum period of time for which an authentication ticket remains valid.

default_certificate_lifetime
The default period of time for which an authentication certificate (ticket-granting ticket) remains valid. A process can request an authentication certificate with a longer lifetime. Note that the maximum lifetime for an authentication certificate cannot exceed the lifetime established by the effective policy for the requesting account.

low_unix_id_person
The lowest UNIX ID that can be assigned to a principal in the registry.

low_unix_id_group
The lowest UNIX ID that can be assigned to a group in the registry.

low_unix_id_org
The lowest UNIX ID that can be assigned to an organization in the registry.

max_unix_id
The maximum UNIX ID that can be used for any item in the registry.

realm A character string naming the cell controlled by this registry.

realm_uuid The UUID of the cell controlled by this registry.

flags Flags include the following:

sec_rgy_prop_readonly
If TRUE, the registry database is read-only.

sec_rgy_prop_auth_cert_unbound
If TRUE, privilege attribute certificates can be generated for use at any site.

sec_rgy_prop_shadow_password
If FALSE, passwords can be distributed over the network. If this flag is TRUE, passwords will be stripped from the returned data to the **sec_rgy_acct_lookup()**, and other calls that return an account's encoded password.

sec_rgy_prop_embedded_unix_id
All registry UUIDs contain embedded UNIX IDs. This implies that the UNIX ID of any registry object cannot be changed, since UUIDs are immutable.

Permissions Required

The **sec_rgy_properties_set_info()** routine requires the **m** (**mgmt_info**) permission on the policy object for which the property information is to be set.

Files

/usr/include/dce/policy.idl
The **idl** file from which **dce/policy.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_not_authorized
The user is not authorized to change the registry properties.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_properties_get_info(3sec)**.

sec_rgy_site_bind

Purpose

Binds to a registry site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_bind(
        unsigned_char_t *site_name,
        sec_rgy_bind_auth_info_t *auth_info,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) specifying the security server to bind to in one of the following forms:

- To bind to an arbitrary security server site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To bind to a specific security server site in a specific cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2)** or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine binds to an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

auth_info A pointer to the **sec_rgy_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and

authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info(3rpc)** reference page.) If the **sec_rgy_bind_auth_info_t** structure specifies authenticated RPC, the caller must have established a valid network identity for this call to succeed.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (''bound to'') the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_bind()** call binds to a registry site at the security level specified by the *auth_info* parameter. The *site_name* parameter identifies the registry to use. If *site_name* is NULL, or a zero-length string, a registry site in the local cell is selected by the client agent.

Note: Like the **sec_rgy_site_bind_query()** routine, this routine binds arbitrarily to either an update or query site. Although update sites can accept queries, query sites cannot accept updates. To specifically select an update site, use **sec_rgy_site_bind_update()**.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_cell_bind(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_site_bind_query

Purpose

Binds to a registry query site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_bind_query(
        unsigned_char_t *site_name,
        sec_rgy_bind_auth_info_t *auth_info,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) specifying the security server to bind to in one of the following forms:

- To bind to an arbitrary security server site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To bind to a specific security server site in a specific cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2)** or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine binds to an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

auth_info A pointer to the **sec_rgy_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and

authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info(3rpc)** reference page.) If the **sec_rgy_bind_auth_info_t** structure specifies authenticated RPC, the caller must have established a valid network identity for this call to succeed.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (''bound to'') the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_bind_query()** routine binds to a registry query site at an arbitrary level of security. A registry query site is a satellite server that operates on a periodically updated copy of the main registry database. To change the registry database, it is necessary to change a registry update site, which then automatically updates its associated query sites. No changes can be made directly to a registry query database.

The *site_name* parameter identifies the query site to use. If *site_name* is NULL, or a zero-length string, a query site in the local cell is selected by the client agent.

The handle for the associated registry server is returned in *context*.

Note: Like **sec_rgy_bind_open()** routine, this routine binds arbitrarily to either an update or query site. Although update sites can accept queries, query sites cannot accept updates. To specifically select an update site, use **sec_rgy_site_bind_update()**.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_bind(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_site_bind_update

Purpose

Binds to a registry update site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_bind_update(
        unsigned_char_t *site_name,
        sec_rgy_bind_auth_info_t *auth_info,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) containing the name of the security server to bind to. Supply this name in any of the following forms:

- To bind to the update site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To start the search for the update site at a specific replica in the replica's cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2**) or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine starts the search at an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

auth_info A pointer to the **sec_rgy_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info(3rpc)** reference page.) If the **sec_rgy_bind_auth_info_t** structure specifies authenticated RPC, the caller must have established a valid network identity for this call to succeed.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (bound to) the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_bind_update()** routine binds to a registry update site. A registry update site is a master server that may control several satellite (query) servers. To change the registry database, it is necessary to change a registry update site, which then automatically updates its associated query sites. No changes can be made directly to a registry query database.

The *site_name* parameter identifies either the cell in which to find the update site or the replica at which to start the search for the update site. If *site_name* is NULL, or a zero-length string, an update site in the local cell is selected by the client agent.

The handle for the associated registry server is returned in *context*. The handle is to an update site. Use this registry context handle in subsequent calls that update or query the the registry database (for example, the **sec_rgy_pgo_add()** or **sec_rgy_acct_lookup()** calls).

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_bind(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_site_binding_get_info

Purpose

Returns information from the registry binding handle

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_binding_get_info(
        sec_rgy_handle_t context,
        unsigned_char_t **cell_name,
        unsigned_char_t **server_name,
        unsigned_char_t **string_binding,
        sec_rgy_bind_auth_info_t *auth_info,
        error_status_t *status);
```

Parameters

Input

context A **sec_rgy_handle_t** variable that contains a registry server handle indicating (bound to) the desired registry site. To obtain information on the default binding handle, initialize *context* to **sec_rgy_default_handle**. A valid login context must be set for the process if *context* is set to **sec_rgy_default_handle**; otherwise the error **sec_under_login_s_no_current_context** is returned.

Output

cell_name The name of the home cell for this registry.

server_name
The name of the node on which the server is resident. This name is either a global name or a network address, depending on the form in which the name was input to the call that bound to the site.

string_binding
A string containing binding information from **sec_rgy_handle_t**.

auth_info A pointer to the **sec_rgy_bind_auth_info_t** structure that identifies the authentication protocol, protection level, and authorization protocol to use in establishing the binding. (See the **rpc_binding_set_auth_info()** routine).

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_binding_get_info()** routine returns the site name and authentication information associated with the *context* parameter. If the context is the default context, the information for the default binding is returned. Passing in a NULL value for any of the output values (except for *status*) will prevent that value from being returned. Memory is allocated for the string returned in the *cell_name*, *server_name*, and *string_binding* parameters. The application calls the **rpc_string_free()** routine to deallocate that memory.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_under_login_s_no_current_context

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_bind(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_site_close

Purpose

Frees the binding handle for a registry server

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_close(
        sec_rgy_handle_t context,
        error_status_t *status);
```

Parameters

Input

context An opaque handle indicating (bound to) a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

Output

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_close()** routine frees the memory occupied by the specified handle and destroys its binding with the registry server.

Notes

A handle cannot be used after it is freed.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_get(3sec)**, **sec_rgy_site_is_readonly(3sec)**, **sec_rgy_site_open(3sec)**, **sec_rgy_site_open_query(3sec)**, **sec_rgy_site_open_update(3sec)**.

sec_rgy_site_get

Purpose

Returns the string representation for a bound registry site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_get(
        sec_rgy_handle_t context,
        unsigned_char_t **site_name,
        error_status_t *status);
```

Parameters

Input

context An opaque handle indicating (bound to) a registry server. Use **sec_rgy_site_open()** to acquire a bound handle. To obtain information on the default binding handle, initialize *context* to **sec_rgy_default_handle**. A valid login context must be set for the process if *context* is set to **sec_rgy_default_handle**; otherwise the error **sec_under_login_s_no_current_context** is returned.

Output

site_name A pointer to a character string (type **unsigned_char_t**) containing the returned name of the registry site associated with *context*, the given registry server handle.

The name is either a global name or a network address, depending on the form in which the name was input to the call that bound to the site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_get()** routine returns the name of the registry site associated with the specified handle. If the handle is the default context, the routine returns the name of the default context's site. Memory is allocated for the string returned in the *site_name* parameter. The application calls the **rpc_string_free()** routine to deallocate that memory.

Notes

To obtain binding information, the use of the **sec_rgy_site_binding_get_info()** call is recommended in place of this call.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_under_login_s_no_current_context

sec_rgy_server_unavailable
The requested registry server is not available.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_open(3sec)**.

sec_rgy_site_is_readonly

Purpose

Checks whether a registry site is read-only

Synopsis

```
#include <dce/binding.h>

boolean32 sec_rgy_site_is_readonly(
        sec_rgy_handle_t context);
```

Parameters

Input

context An opaque handle indicating (bound to) a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

Description

The **sec_rgy_site_is_readonly()** routine checks whether the registry site associated with the specified handle is a query site or an update site. A query site is a read-only replica of a master registry database. The update site accepts changes to the registry database, and duplicates the changes in its associated query sites.

Return Values

The routine returns

- TRUE, if the registry site is read-only or if there was an error using the specified handle
- FALSE, if the registry site is an update site

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_open(3sec)**, **sec_rgy_site_open_query(3sec)**.

sec_rgy_site_open

Purpose

Binds to a registry site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_open(
        unsigned_char_t *site_name,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) specifying the security server to bind to in one of the following forms:

- To bind to an arbitrary security server site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To bind to a specific security server site in a specific cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2**) or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine binds to an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (bound to) the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_open()** routine binds to a registry site at the level of security specified in the **rpc_binding_set_auth_info()** call. The *site_name* parameter identifies the registry to use. If *site_name* is NULL, or a zero-length string, a registry site in the local cell is selected by the client agent. The caller must have established a valid network identity for this call to succeed.

Note: To bind to a registry site, the use of the **sec_rgy_site_bind()** call is recommended in place of this call.

Like **sec_rgy_site_open_query()** routine, this routine binds arbitrarily to either an update or query site. Although update sites can accept queries, query sites cannot accept updates. To specifically select an update site, use **sec_rgy_site_open_update()**.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The requested registry server is not available.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_close(3sec)**, **sec_rgy_site_is_readonly(3sec)**, **sec_rgy_site_open_query(3sec)**, **sec_rgy_site_open_update(3sec)**.

sec_rgy_site_open_query

Purpose

Binds to a registry query site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_open_query(
        unsigned_char_t *site_name,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) specifying the security server to bind to in one of the following forms:

- To bind to an arbitrary security server site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To bind to a specific security server site in a specific cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2**) or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine binds to an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (bound to) the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_open_query()** routine binds to a registry query site. A registry query site is a satellite server that operates on a periodically updated copy of the main registry database. To change the registry database, it is necessary to change a registry update site, which then automatically updates its associated query sites. No changes can be made directly to a registry query database.

The *site_name* parameter identifies the query site to use. If *site_name* is NULL, or a zero-length string, a query site in the local cell is selected by the client agent.

The handle for the associated registry server is returned in *context*.

The caller must have established a valid network identity for this call to succeed.

Note: To bind to a registry query site, the use of the **sec_rgy_site_bind_query()** call is recommended in place of this call.

Like **sec_rgy_site_open()** routine, this routine binds arbitrarily to either an update or query site. Although update sites can accept queries, query sites cannot accept updates. To specifically select an update site, use **sec_rgy_site_open_update()**.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_close(3sec)**, **sec_rgy_site_get(3sec)**, **sec_rgy_site_is_readonly(3sec)**, **sec_rgy_site_open(3sec)**, **sec_rgy_site_open_update(3sec)**.

sec_rgy_site_open_update

Purpose

Binds to a registry update site

Synopsis

```
#include <dce/binding.h>

void sec_rgy_site_open_update(
        unsigned_char_t *site_name,
        sec_rgy_handle_t *context,
        error_status_t *status);
```

Parameters

Input

site_name A character string (type **unsigned_char_t**) specifying the security server to bind to in one of the following forms:

- To bind to the update site in a named cell, specify a cell name (for example, **/.../r_d.com**) or **/.:** for the local cell.
- To start the search for the update site at a specific replica in the replica's cell, specify either the cell name and the server name (for example, **/.../r_d.com/subsys/dce/sec/rs_server_250_2**) or the server's network address (for example, **ncadg_ip_udp:15.22.144.248**). If the server name is not valid, the routine binds to an arbitrary security site in the named cell.

Note that the routine ignores anything after the cell name that does not refer to an item in the Cell Directory Service (CDS) namespace. If the specified CDS namespace item does not resolve to a security server, the call fails.

Output

context A pointer to a **sec_rgy_handle_t** variable. Upon return, this contains a registry server handle indicating (bound to) the desired registry site.

status A pointer to the completion status. On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_site_open_update()** routine binds to a registry update site. A registry update site is a master server that may control several satellite (query) servers. To change the registry database, it is necessary to change a registry update site, which then automatically updates its associated query sites. No changes can be made directly to a registry query database.

The *site_name* parameter identifies either the cell in which to find the update site or the replica at which to start the search for the update site. If *site_name* is NULL, or a zero-length string, an update site in the local cell is selected by the client agent.

The handle for the associated registry server is returned in *context*. The handle is to an update site. Use this registry context handle in subsequent calls that update or query the the registry database (for example, the **sec_rgy_pgo_add()** or **sec_rgy_acct_lookup()** calls). The caller must have established a valid network identity for this call to succeed.

Note: To bind to a registry update site, the use of the **sec_rgy_site_bind_update()** call is recommended in place of this call.

Files

/usr/include/dce/binding.idl
The **idl** file from which **dce/binding.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_login_s_no_current_context
The caller does not have a valid network login context.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**, **sec_rgy_site_close(3sec)**, **sec_rgy_site_get(3sec)**, **sec_rgy_site_is_readonly(3sec)**, **sec_rgy_site_open(3sec)**, **sec_rgy_site_open_query(3sec)**.

sec_rgy_unix_getgrgid

Purpose

Returns a UNIX style group entry for the account matching the specified group ID

Synopsis

```
#include <dce/rgynbase.h>

void sec_rgy_unix_getgrgid(
        sec_rgy_handle_t context,
        signed32 gid,
        signed32 max_number,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_unix_group_t *group_entry,
        signed32 *number_members,
        sec_rgy_member_t member_list[ ],
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

gid A 32-bit integer specifying the group ID to match.

max_number The maximum number of members to be returned by the call. This must be no larger than the allocated size of the *member_list[ ]* array.

Input/Output

item_cursor

An opaque pointer indicating a specific principal, group, and organization (PGO) item entry in the registry database. The **sec_rgy_unix_getgrgid()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine

returns **sec_rgy_no_more_entries**. Use **sec_rgy_cursor_reset()** to refresh the cursor.

Output

group_entry A UNIX style group entry structure returned with information about the account matching *gid*.

number_members
A signed 32-bit integer containing the total number of member names returned in the *member_list[ ]* array.

member_list[ ]
An array of character strings to receive the returned member names. The size allocated for the array is given by *max_number*. If this value is less than the total number of members in the membership list, multiple calls must be made to return all of the members.

status On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_unix_getgrgid()** routine returns the next UNIX group structure that matches the input UNIX group ID. The structure is in the following form:

```
typedef struct {
   sec_rgy_name_t name;
   signed32 gid;
   sec_rgy_member_buf_t members;
}       sec_rgy_unix_group_t;
```

The structure includes the following:

- The name of the group
- The group's UNIX ID
- A string containing the names of the group members. This string is limited in size by the size of the **sec_rgy_member_buf_t** type defined in **rgynbase.idl**.

The routine also returns an array of member names, limited in size by the *number_members* parameter.

This call is supplied in source code form.

Files

/usr/include/dce/rgynbase.idl
The **idl** file from which **dce/rgybase.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_nomore_entries
The end of the list of entries has been reached.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_unix_getgrnam

Purpose

Returns a UNIX style group entry for the account matching the specified group name

Synopsis

```
#include <dce/rgynbase.h>

void sec_rgy_unix_getgrnam(
        sec_rgy_handle_t context,
        sec_rgy_name_t name,
        signed32 name_length,
        signed32 max_num_members,
        sec_rgy_cursor_t item_cursor,
        sec_rgy_unix_group_t group_entry,
        signed32 number_members,
        sec_rgy_member_t member_list[ ],
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

name A character string (of type **sec_rgy_name_t**) specifying the group name to be matched.

name_length
An signed 32-bit integer specifying the length of *name* in characters.

max_num_members
The maximum number of members to be returned by the call. This must be no larger than the allocated size of the *member_list[ ]* array.

Input/Output

item_cursor
: An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_unix_getgrnam()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns **sec_rgy_no_more_entries**. Use **sec_rgy_cursor_reset()** to refresh the cursor.

Output

group_entry
: A UNIX style group entry structure returned with information about the account matching *name*.

number_members
: An signed 32-bit integer containing the total number of member names returned in the *member_list[ ]* array.

member_list[ ]
: An array of character strings to receive the returned member names. The size allocated for the array is given by *max_number*. If this value is less than the total number of members in the membership list, multiple calls must be made to return all of the members.

status
: On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_unix_getgrnam()** routine looks up the next group entry in the registry that matches the input group name and returns the corresponding UNIX style group structure. The structure is in the following form:

```
typedef struct {
    sec_rgy_name_t name;
    signed32 gid;
    sec_rgy_member_buf_t members;
}           sec_rgy_unix_group_t;
```

The structure includes the following:

- The name of the group.
- The group's UNIX ID.
- A string containing the names of the group members. This string is limited in size by the size of the **sec_rgy_member_buf_t** type defined in **rgynbase.idl**.

The routine also returns an array of member names, limited in size by the *number_members* parameter. Note that the array contains only the names explicitly specified as members of the group. A principal that was made a member of the group because that group was assigned as the principal's primary group will not appear in the array.

This call is provided in source code form.

Files

/usr/include/dce/rgynbase.idl
The **idl** file from which **dce/rgybase.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The end of the list of entries has been reached.

sec_rgy bad_data
The name supplied as input was too long.

error_status_ok
The call was successful.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_unix_getpwnam

Purpose

Returns a UNIX style passwd entry for account matching the specified name

Synopsis

```
#include <dce/rgynbase.h>

void sec_rgy_unix_getpwnam (
        sec_rgy_handle_t context,
        sec_rgy_name_t name,
        unsigned32 name_len,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_unix_passwd_t *passwd_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open** to acquire a bound handle.

name A character string (of type **sec_rgy_name_t**) containing the name of the person, group, or organization whose name entry is desired.

name_len A 32-bit integer representing the length of the *name* in characters.

Input/Output

item_cursor

An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_unix_getpwnam** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns **sec_rgy_no_more_entries**. Use **sec_rgy_cursor_reset** to refresh the cursor.

Output

passwd_entry
: A UNIX style passwd structure returned with information about the account matching *name*.

status
: On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_unix_getpwnam** routine returns the next UNIX passwd structure that matches the input name. The structure is in the following form:

```
typedef struct {
    sec_rgy_unix_login_name_t name;
    sec_rgy_unix_passwd_buf_t passwd;
    signed32 uid;
    signed32 gid;
    signed32 oid;
    sec_rgy_unix_gecos_t gecos;
    sec_rgy_pname_t homedir;
    sec_rgy_pname_t shell;
}           sec_rgy_unix_passwd_t;
```

The structure includes the following:

- The account’s login name.
- The account’s password.
- The account’s UNIX ID.
- The UNIX ID of group and organization associated with the account.
- The account’s GECOS information.
- The account’s home directory.
- The account’s login shell

This call is provided in source code form.

Files

/usr/include/dce/rgynbase.idl
The **idl** file from which **rgynbase.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy bad_data
The name supplied as input was too long.

error_status_ok
The call was successful.

sec_rgy_no_more_entries
The end of the list of entries has been reached.

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_unix_getpwuid

Purpose

Returns a UNIX style **passwd** entry for the account matching the specified UID

Synopsis

```
#include <dce/rgynbase.h>

void sec_rgy_unix_getpwuid(
        sec_rgy_handle_t context,
        signed32 uid,
        sec_rgy_cursor_t *item_cursor,
        sec_rgy_unix_passwd_t *passwd_entry,
        error_status_t *status);
```

Parameters

Input

context An opaque handle bound to a registry server. Use **sec_rgy_site_open()** to acquire a bound handle.

uid A 32-bit integer UNIX ID.

Input/Output

item_cursor

An opaque pointer indicating a specific PGO item entry in the registry database. The **sec_rgy_unix_getpwuid()** routine returns the PGO item indicated by *item_cursor*, and advances the cursor to point to the next item in the database. When the end of the list of entries is reached, the routine returns **sec_rgy_no_more_entries**. Use **sec_rgy_cursor_reset()** to refresh the cursor.

Output

passwd_entry

A UNIX style password structure returned with information about the account matching *uid*.

status On successful completion, the routine returns **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_unix_getpwuid()** routine looks up the next **passwd** entry in the registry that matches the input UNIX ID and returns the corresponding **sec_rgy_passwd** structure. The structure is in the following form:

```
typedef struct {
    sec_rgy_unix_login_name_t name;
    sec_rgy_unix_passwd_buf_t passwd;
    signed32 Vuid;
    signed32 Vgid;
    signed32 oid;
    sec_rgy_unix_gecos_t gecos;
    sec_rgy_pname_t homedir;
    sec_rgy_pname_t shell;
}           sec_rgy_unix_passwd_t;
```

The structure includes the following:

- The account’s login name.
- The account’s password.
- The account’s UNIX ID.
- The UNIX ID of group and organization associated with the account.
- The account’s GECOS information.
- The account’s home directory.
- The account’s login shell

Files

/usr/include/dce/rgynbase.idl
The **idl** file from which **dce/rgynbase.h** was derived.

This call is provided in source code form.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_no_more_entries
The end of the list of entries has been reached.

sec_rgy_server_unavailable
The DCE registry server is unavailable.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

sec_rgy_wait_until_consistent

Purpose

Blocks the caller while prior updates are propagated to the registry replicas

Synopsis

```
#include <dce/misc.h>

boolean32 sec_rgy_wait_until_consistent(
        sec_rgy_handle_t context,
        error_status_t *status);
```

Parameters

Input

context The registry server handle associated with the master registry.

Output

status A pointer to the completion status. On successful completion, *status* is assigned **error_status_ok**. Otherwise, it returns an error.

Description

The **sec_rgy_wait_until_consistent()** routine blocks callers until all prior updates to the master registry have been propagated to all active registry replicas.

Return Values

The routine returns TRUE when all active replicas have received the prior updates. It returns FALSE if at least one replica did not receive the updates.

Files

/usr/include/dce/misc.idl
The **idl** file from which **dce/misc.h** was derived.

Errors

The following describes a partial list of errors that might be returned. Refer to the *DCE 1.2.2 Problem Determination Guide* for complete descriptions of all error messages.

sec_rgy_server_unavailable
The server for the master registry is not available.

sec_rgy_read_only
Either the master site is in maintenance mode or the site associated with the handle is a read-only (query) site.

error_status_ok
The call was successful.

Related Information

Functions: **sec_intro(3sec)**.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y