Specification: Certificate Management Services Application Programming Interface

Issue 2.0

Abstract

The Certificate Management Services Application Programming Interface specification defines the interface between a client-end Security Application and a certificate management infrastructure for management and distribution of public key certificates and public/private key pairs.

Comments

Please provide any comments or suggestions for this specification to Tim Moses at Nortel Technologies, PO Box 3511, Station C, Ottawa, Ontario, Canada, K1Y 4H7. Tel (613) 763 2694, Fax: (613) 765 3520, Internet: timmoses@bnr.ca.

Revision History

Issue	Date	Parameters of changes in this issue
0.1	11 Aug 1995	First draft issue
0.2	18 Aug 1995	A set of search directory functions were added. Update key calls were reduced to a single step. Functions were grouped into functional categories. Changes were made to allow users to supply encryption keys for certification and to request that the CA provide a signature key.
0.3	25 Aug 1995	A set of functions were added to allow path processing functions to be linked into the CMS at some future time. An introduction was added. Status codes were added.
0.4	1 Sep 1995	Added “oldCertificate” to the update key calls
0.5	22 Sep 1995	Revised Introductory Material. Added the CMSEraseKeyHistory, CMSGetCertificate and the CMSKeyStatus calls. Deleted the Path Processor functions. Grouped function parameters into a smaller number of structured parameters. Added return codes to support a Cryptoki crypto library.
1.0	6 Oct 1995	Introduced a ‘quality of certificate’ parameter. Provided a mechanism for environments in which the CA is not available ‘on-line’. Modified the initialization function to initialize a single key pair. Introduced a new name list, called ‘ValidNameList’, to contain just those names whose certificates have been validated. Added functions to return individual fields from a certificate. Modified the certificate path list to contain just the subject DN and the serial number (as opposed to the whole certificate). In the case of failure to verify, the list will only contain details of those certificates that failed.
2.0	2 Aug 1996	Moved C language bindings to an appendix. Modified the certificate verification model and the associated calls. Added name conversion functions to allow applications to deal in application-specific name forms. **Add more detail**

Table Of Contents

1. INTRODUCTION ---------- 1

1.1 Abbreviations ---------- 1

1.2 Purpose ---------- 1

1.3 Field of Application ---------- 2

1.4 Infrastructure Architecture ---------- 2

1.5 Keys and Certificates ---------- 3
1.5.1 Types of Key ---------- 3
1.5.2 Key Life-cycle ---------- 4

1.6 List Objects ---------- 7
1.6.1 SearchResultsNameList ---------- 7
1.6.2 RecoveredKeyList ---------- 8

1.7 Memory Ownership ---------- 8

1.8 Name Conversion Functions ---------- 9

1.9 CMS Utilities ---------- 9

1.10 Requirements ---------- 9

2. INTERFACE SPECIFICATION ---------- 10

2.1 Structured Parameters ---------- 10
2.1.1 Key Flow ---------- 10
2.1.2 Certificate Attributes ---------- 10
2.1.3 Registration Form ---------- 10
2.1.4 Key Pair ---------- 10

2.2 Context-Related Functions ---------- 11
2.2.1 CMSLogin ---------- 11
2.2.2 CMSLogout ---------- 13

2.3 Registration Authority Functions ---------- 14
2.3.1 CMSCertificationRequest ---------- 14
2.3.2 CMSCertificationResponse ---------- 15
2.3.3 CMSRevokeACertificate ---------- 16

2.4 End-user Registration Functions ---------- 17
2.4.1 CMSRequestRegistrationForm ---------- 17
2.4.2 CMSRetrieveRegistrationForm ---------- 18
2.4.3 CMSRegistrationRequest ---------- 19
2.4.4 CMSRegistrationResponse ---------- 20

2.5 Certification Functions ---- 21
2.5.1 CMSRequestMyCertificate ---- 21
2.5.2 CMSRetrieveMyCertificate ---- 23

2.6 Key Update Functions ---- 25
2.6.1 CMSKeyStatus ---- 25
2.6.2 CMSRequestKeyUpdate ---- 26
2.6.3 CMSRetrieveUpdatedKey ---- 28

2.7 Key Recovery Functions ---- 30
2.7.1 CMSRequestKeyRecovery ---- 30
2.7.2 CMSRetrieveRecoveredKeys ---- 32
2.7.3 CMSNumberOfRecoveredKeys ---- 34
2.7.4 CMSGetRecoveredCertificate ---- 35
2.7.5 CMSGetRecoveredKey ---- 36
2.7.6 CMSEraseKeyHistory ---- 37

2.8 End-user Self-Revocation Functions ---- 38
2.8.1 CMSRevokeMyCertificate ---- 38

2.9 Name Resolution Functions ---- 39
2.9.1 CMSSearchDirectory ---- 39
2.9.2 CMSNumberOfNames ---- 41
2.9.3 CMSGetUniqueName ---- 42
2.9.4 CMSNumberOfAttributes ---- 43
2.9.5 CMSGetAttribute ---- 44
2.9.6 CMSNumberOfAttributeValues ---- 45
2.9.7 CMSGetAttributeValue ---- 46
2.9.8 CMSResetNameList ---- 47

2.10 Certificate Verification Functions ---- 48
2.10.1 CMSValidateCertificate ---- 48
2.10.2 CMSValidateNextLink ---- 51
2.10.3 CMSGetCertificateAttribute ---- 52

2.11 Utility Functions ---- 54
2.11.1 CMSReleaseBuffer ---- 54
2.11.2 CMSReleaseName ---- 55
2.11.3 CMSReleaseKey ---- 56
2.11.4 CMSGetLogString ---- 57
2.11.5 CMSLogToString ---- 58
2.11.6 CMSQueryLogWarning ---- 59
2.11.7 CMSQueryVersionNumber ---- 60
2.11.8 Name Conversion Functions ---- 61

3. STATUS CODES ---- 62

3.1 Error Codes ---- 62

3.2 Warning Codes ---- 64

3.3 Normal Operation Codes ---- 64

4. CERTIFICATE ATTRIBUTE OBJECT IDENTIFIERS---------- 65

5. REFERENCES---------- 70

ANNEX A - API PROFILE FOR ENTRUST/CMS V1.0----------68

1. Introduction

1.1 Abbreviations

API	Application Programming Interface
ARL	Authority Revocation List
BER	Basic Encoding Rules for the ASN.1 syntax
CA	Certification Authority
CRL	Certificate Revocation List
CMS	Certificate Management Services
DIT	Directory Information Tree
DSA	Directory System Agent
I&A	Identification and Authentication
KRC	Key Recovery Centre
OID	Object Identifier
PIN	Personal Identification Number
RA	Registration Authority
RSA	Rivest Shamir and Adelman

1.2 Purpose

There is a growing need to secure information flows which span organizational boundaries. In addition, there is a large installed base of information technology with no significant degree of uniformity: components of the systems are supplied by a wide range of vendors and applications exhibit a wide range of security sensitivity. Furthermore, security and infrastructure requirements are complex and rapidly evolving. Therefore, the implementation of suitable security infrastructures for the installed base of information systems and existing applications may lead to high capital and operating costs.

The standardization of an interface between Security Applications and a security infrastructure will lead to greater choice and less duplication in the implementation of security infrastructures, which will, in turn, translate into lower end-user costs. The API defined in this specification provides an interface to a security infrastructure which is

capable of satisfying the requirements of heterogeneous distributed systems operating across multiple policy domains with platforms supplied by different vendors.

1.3 Field of Application

The infrastructure addressed by this API is based upon the X.500 distributed directory (***see Ref. 3***). It satisfies the requirements of all end-user and Certification Authority public/private key life cycle phases. It uses the X.509 standard for public key certificates. However, for simpler applications the semantics of the certificate may be opaque to the Security Application. Compliant infrastructures may use any of the available versions of the X.509 certificate, but the features offered will be dependent upon the chosen version.

1.4 Infrastructure Architecture

The operating context of the CMS-API is shown in Figure 1. This context has five main components: the Security Application, the Certificate Management Services Client, the Crypto Library, the Certification Authority and the Certificate and CRL Repository. **The RA must be added to the diagram**.

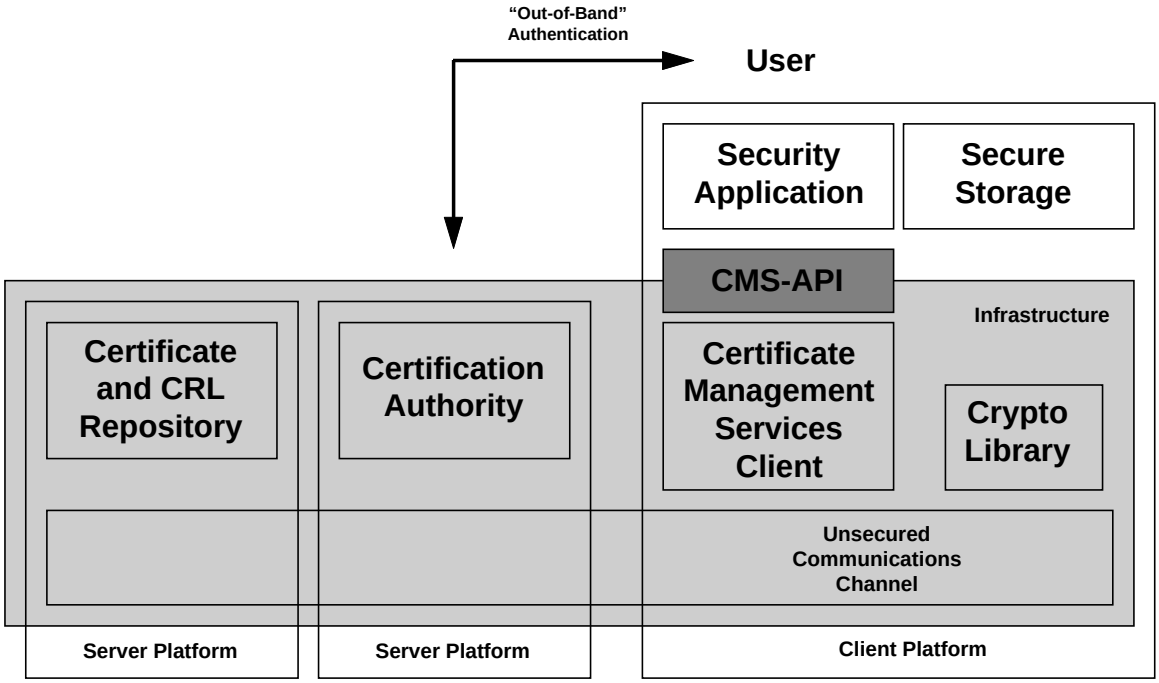


Figure 1 - CMS-API Operating Context

The function of the Security Application is to deliver security services to human users or information processes. These services may include: confidentiality, integrity, authenticity and non-repudiation services, such as Proof of Origin, Proof of Delivery and Proof of Transport. In order to do this in a widely distributed information system, the Security Application needs access to certificate management services. These are

delivered through the CMS-API by the Certificate Management Services (CMS) Client. The CMS can be viewed as the extension of the infrastructure on the client platform.

The transport of cryptographic keys, in the form of certificates, is achieved by means of the repository. And the authenticity and integrity of those keys is assured by the Certification Authority (CA). The CMS Client, CA and Repository communicate over an unsecured communication channel, such as an on-line TCP/IP network.

The Security Application and the CMS Client run on the same platform, so there are no requirements for confidentiality, integrity or authenticity mechanisms in the communications between them. Looked at another way, the Security Application trusts the CMS Client to act on its behalf with respect to key and certificate management functions.

The CMS Client uses cryptographic mechanisms, such as encryption and digital signature, in the performance of its functions. These are provided by the Crypto Library which may be a software process built into the CMS Client, or it may be a software library or token provided by the Security Application.

1.5 Keys and Certificates

The function of the CMS is to manage and deliver end-user cryptographic keys in a secure manner.

1.5.1 Types of Key

All the keys managed by the CMS are public (or asymmetric) key pairs. Any symmetric keys used by the Security Application in the encryption of user data fall outside the scope of the infrastructure.

All certificates are produced from their public key by the infrastructure. However, the key pairs may either be supplied by the infrastructure or by the Security Application. When the key pairs are supplied by the infrastructure, it is its responsibility to ensure the soundness of the key pair for the algorithm in which it is to be used. But when keys are supplied by the Security Application, it is the responsibility of the Security Application to ensure the soundness of the key pairs.

Either the infrastructure or the Security Application may post the end-user's certificates to the repository. But the repository access schema must have been set to allow the appropriate entity write privilege for the end-user's certificate attribute, and the necessary authentication information must have been established between the repository and that entity. Interfaces that allow posting of the certificate by the Security Application are outside the scope of this specification.

The user may have certificates issued by more than one CA. Therefore, the directory must be capable of accommodating multi-valued attributes and managing those values separately.

1.5.2 Key Life-cycle

The infrastructure provides support for all phases of the key life-cycle, including Registration, Certification, Recovery, Update and the Active phase. Registration and Deregistration exchanges between a Registration Authority and the Certification Authority are included in the scope of this specification. **Consider whether interfaces between the RA and end-user at the RA end should be included.**

1.5.2.1 Registration Phase

The Registration Phase establishes a shared secret between the end-user and the CA as a precursor to the certification phase. It involves exchanges between the end-user and the Registration Authority (RA). The responsibility of the RA is to qualify the end-user as a certificate owner, including (potentially) establishing its identity. The Security Application uses **CMSRequestRegistrationForm** and **CMSRetrieveRegistrationForm** to obtain a registration form from the Registration Authority. Communication with the RA may be 'on-line', in which case the retrieve call may immediately follow the request call. If the communication with the CA is via a 'store-and-forward' network, then the request call will provide an estimate of the time until the retrieve call can be successfully completed.

1.5.2.2 Certification Phase

The certification phase establishes a certificate for the end-user. Authentication information may be established 'out-of-band' or as a result of the registration phase. The semantics of the authentication information are opaque to the Security Application, which simply includes it in the appropriate function call parameter. The user must ensure that this information is communicated and destroyed in a confidential manner. In the case of CAs that support multiple policies, the Security Application has the ability to select a certificate in conformance with any one of the available policies. The CA may reject the request if it does not conform with applicable details of the policy.

The security application uses the CMS functions **CMSRequestMyCertificate** and **CMSRetrieveMyCertificate** to obtain initial certificates from the infrastructure and (optionally) to have them posted to the repository.

1.5.2.3 Update Phase

Periodically, keys and their certificates have to be replaced. This has the effect of restricting the amount of information protected by any one key, thereby preventing certain forms of cryptanalytic attack which are based upon the analysis of many encrypted or signed information objects. Replacement of a key pair must be initiated by the Security Application, and the API provides the **CMSKeyStatus** function to help the Security Application determine when a certificate should be replaced.

The Security Application uses the CMS functions **CMSRequestKeyUpdate** and **CMSRetrieveUpdatedKey** to replace a certificate for any one of its key pairs.

Identification and Authentication (I&A) of the user is required for the purpose of key update. Identification relies upon the user's unique name, as encoded in its certificate. If the CMS Client uses the crypto library which is built into the CMS Client, then authentication relies upon an authentication token supplied to the Security Application during the most recent **CMSRetrieveMyCertificate**, **CMSRetrieveRecoveredKeys** or **CMSRetrieveUpdatedKey** call. The semantics of the authentication token are opaque to the Security Application, which merely has to afford it confidential storage. If the CMS Client uses the Crypto Library contained in a token, then the authentication information may be used as the token PIN. **Currently the interface only supports the update of the 'key'. i.e. if a new certificate is requested, it must contain a new key. Consideration should be given to supporting update of a certificate only, keeping the key the same.**

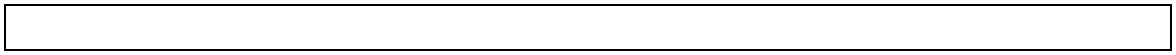
1.5.2.4 Recovery Phase

In the event that the Security Application cannot access its private key, perhaps due to a disk failure, it will be necessary to obtain new private key pairs and their associated certificates, and to recover the history of its private keys. The CMS-API includes a call that allows the Security Application to recover any previous private key held by the CA, in an authentic and confidential manner, placing them in a local list. This is achieved through use of the functions **CMSRequestKeyRecovery** and **CMSRetrieveRecoveredKeys.** Section 1.6.2 describes calls used to process the list of recovered keys. **CMSEraseKeyHistory** should be used to erase the CMS memory image of the list of keys recovered by the **CMSRetrieveRecoveredKeys** function call.

1.5.2.5 Active Phase

In the active phase of the key life-cycle, the CMS provides support for the retrieval and verification of public keys, by means of certificates issued and distributed by the infrastructure. There are two stages in the process of delivering verified public keys (see Figure 2). These stages are:

- Name Resolution (which may include Certificate Retrieval), and
- Certificate Verification (which may also include Certificate Retrieval).



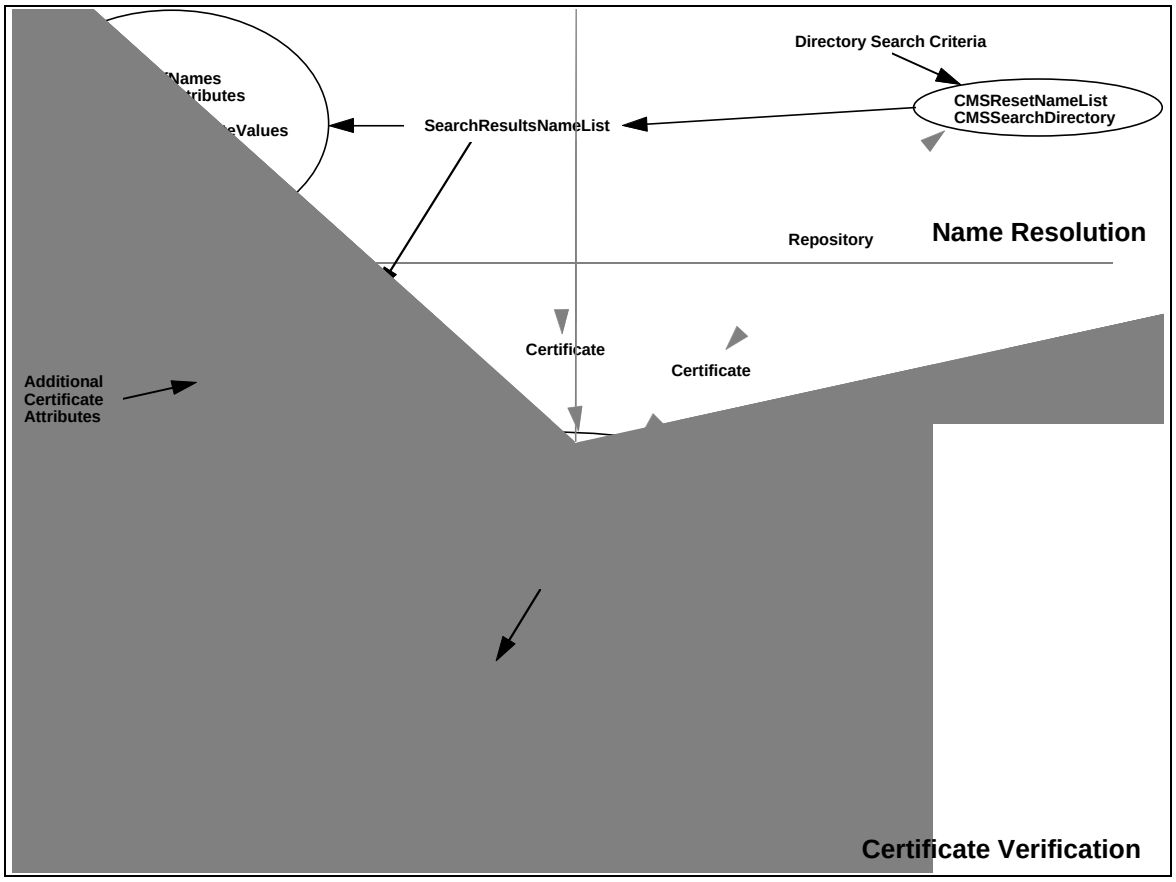


Figure 2 - The Active Phase of the Key Life-Cycle

There are three possible cases that must be considered:

1. The user knows something about the entities for which it requires verified public keys, but not their Unique Names (this situation may arise when a user needs to send an encrypted message to entities with which it has had no previous contact);

2. The user knows the Unique Names of the entities for which it requires verified public keys (this situation may arise when a user needs to send an encrypted message to entities with which it has previously exchanged encrypted messages);

3. The user has a certificate from which it must obtain the verified public key (this situation may arise when the user receives a signed message, including a certificate, from another user).

In the first case, both stages of the public key delivery process must be executed sequentially. In the other two cases, only the second stage of the process must be executed.

The main data objects involved in the public key delivery process are the repository, UsersNameList and the CertificateAttributes list. Upon completion of the Name Resolution stage of the process, the UsersNameList contains the Unique Names and

(optionally) the certificates of the intended recipients. Upon completion of the Certificate Verification stage, the CertificateAttributes list contains verified attributes from the certificate. These attributes include the public key.

One other data object is important to the Certificate Verification process. The SearchResultsNameList is used in the Name Resolution stage and it contains the Unique Names and selected attributes retrieved from the directory. The Security Application may use this data object to retrieve user attributes other than those contained in the certificate.

The functions **CMSSearchDirectory, CMSResetNameList, CMSNumberOfAttributes, CMSGetAttribute, CMSNumberOfAttributeValues and CMSGetUniqueName** can be used to resolve Unique Names and build the UsersNameList with resolved names and (optionally) certificates.

The functions **CMSValidateCertificate and CMSValidateNextLink** can be used to verify a certificate. Certificate criteria, including the Unique Name may be supplied by the application. The certificate may be supplied by the application or it may be obtained from the UserNameList or from the repository.

The function **CMSGetCertificateAttribute** can be used to extract selected attributes (such as the public key) from the verified certificate.

1.6 List Objects

There are two lists maintained by the CMS which are visible to the Security Application. These are:

The SearchResultsNameList,

The RecoveredKeyList and

These are described in the following sections.

1.6.1 SearchResultsNameList

The SearchResultsNameList contains the results of the most recent **CMSSearchDirectory** operation. Its structure is as follows[1]:

```
SearchResultsNameList ::= SEQUENCE OF { SearchResultsNameListEntry }

SearchResultsNameListEntry ::= SEQUENCE {
        uniqueName   Name,
        attributes   Attributes }
```

[1] ASN.1 syntax is used to define the list structure, but the actual encodings of the lists in the implementation is at the discretion of the implementor.

```
Attributes ::= SEQUENCE OF {Attribute }

Attribute ::= SEQUENCE {
        attributeType   printableString,
        values          Values }
Values ::= SEQUENCE OF { Value }

Value ::= OCTET STRING;
```

The following functions are used to maintain this list.

CMSResetNameList removes all entries from the list.

1.6.2 RecoveredKeyList

The RecoveredKeyList contains the result of the most recent **CMSRetrieveRecoveredKeys** operation. Its structure is as follows:

```
RecoveredKeyList ::= SEQUENCE OF { RecoveredKey }

RecoveredKey ::= SEQUENCE {
        certificate             Certificate,
        privateKey              OCTET STRING }
```

It is valid following a call to **CMSRetrieveRecoveredKeys**.

There are four function calls available to maintain the RecoveredKeyList.

CMSGetNumberOfRecoveredKeys returns the number of entries in the list.

CMSGetRecoveredKey returns the private key associated with an index in the list.

CMSGetRecoveredCertificate returns the certificate associated with an index in the list.

The Security Application should make a call to **CMSEraseKeyHistory** as soon as it has completed the key recovery process, in order to ensure that the memory image of the decryption keys is not maintained any longer than is absolutely necessary.

1.7 Memory Ownership

The sizes of some information objects returned by the CMS Client functions are not known, a priori, by the Security Application. Therefore, the CMS Client deals with their retrieval and storage. The Security Application can examine their size and decide whether or not to copy them into memory that it controls. This raises the requirement to form a context for the CMS services.

There are two function calls which deal with the establishment and release of a context:

CMSLogin establishes a context, returning a context handle which the Security Application uses in subsequent CMS calls.

CMSLogout releases the context and de-allocates any memory allocated by the context.

1.8 Name Conversion Functions

Name conversion functions from the GSS-API are included to permit applications to operate in their natural name space.

1.9 CMS Utilities

The CMS-API supports seven utility function calls.

CMSReleaseBuffer releases memory allocated to a buffer.

CMSReleaseName releases memory allocated to a name.

CMSReleaseKey releases memory allocated to a key pair.

CMSQueryVersionNumber returns the version number of the CMS.

CMSGetLogString returns a string explaining a specified CMSLog code in its argument list.

CMSLogToString returns a string explaining a CMSLog code as its return value.

CMSQueryLogWarning indicates whether the supplied CMSLog code represents a warning or an error.

1.10 Requirements

The CMS-API has been defined on the assumption that the following requirements are satisfied by other components of the architecture.

A network time service is available to the Security Application or to the CMS Client.

A repository which can hold multiple certificates for each directory entry.

The repository access control schema must be set so that the CA and/or the end-user can write the certificate attribute.

2. Interface Specification

2.1 Structured Parameters

2.1.1 Key Flow

The key flow parameter indicates where keys should be generated and which entity is responsible for publishing them in the repository.

2.1.2 Certificate Attributes

The certificateAttributes parameter contains a list of attributes for a certificate. This structure is used in a number of ways. It is used in the CMSRequestMyCertificate function to specify attribtues of the certificate requested by the end-entity from the CA. It is used in the CMSValidateCertificate function to specify required attributes of the certificates in the certificate chain that must be verified by the implementation. And it is used in the CMSValidateCertificate function to return verified attributes of the validated certificate.

2.1.3 Registration Form

The registrationForm parameter is used to convey the titles of the fields of the registration form from the RA to the end-entity, and to convey the completed entries from the end-entity to the RA.

2.1.4 Key Pair

The key pair parameter contains all information relevant to a key pair, including the corresponding certificate.

2.2 Context-Related Functions

This section contains functions related to maintaining a context for CMS operations.

2.2.1 CMSLogin

This function creates a context associated with a user. Memory required by the CMS Client will be associated with the context and de-allocated as a result of a subsequent call to CMSLogout.

2.2.1.1 Parameters

cMSVersionNumber (Security Application -> CMS)

cMSVersionNumber shall have the value "2.0". Any other value will cause the function to return an error status code.

cryptoLibrary (Security Application -> CMS)

Indicates which crypto library to use for protecting the confidentiality, integrity and authenticity of the communications with the CA.

cryptoLibraryPIN (Security Application -> CMS)

The PIN used to access the cryptoLibrary.

initializationFile (Security Application -> CMS)

The path to the file used for initialization. This may contain information required by the implementation to perform its function, such as the IP address of the RA, CA, KRC and the directory server.

userId (Security Application -> CMS)

The identity of the user. The implementation may use this to identify credentials maintained by the crypto library.

context (CMS -> Security Application)

The CMS will return a context handle for the CMS session. This will be empty if cMSVersionNumber is not "2.0".

2.2.1.2 Returned Values

CMS_CryptoLibraryNotAvailable

CMS_CryptoLibraryNotSupported

CMS_CryptoLibraryPINIncorrect

CMS_MaximumNumberOfOpenContextsExceeded

CMS_OK

CMS_VersionNumberNotSupported

2.2.2 CMSLogout

The CMS de-allocates all memory objects associated with the context.

2.2.2.1 Parameters

context (Security Application -> CMS)

The context handle for the CMS session which is to be closed. See Section 2.2.1.1 above.

2.2.2.2 Returned Values

CMS_ContextNotRecognized

CMS_OK

2.3 Registration Authority Functions

This set of functions may be used by an application acting as a Registration Authority.

2.3.1 CMSCertificationRequest

CMSCertificationRequest is used by a Registration Authority to prepare a Certification Authority to issue a certificate for an end-entity. The CA response can be obtained by making a call to CMSCertificationResponse. The actual issuance of the corresponding certificate takes place as a result of a call to CMSRequestMyCertificate by the end-entity. The CA plays no part in the assessment of the end-entity's qualifications: it relies entirely upon the RA for this.

2.3.1.1 Parameters

context

See Section 2.2.2.1 above.

certificateAttributes (Security Application -> CMS)

A list of attributes for the approved certificate. The 'privileges' that will be encoded in the certificate by the CA should not exceed those authorized by the RA which are encoded in this list.

requestHandle (CMS -> Security Application)

The handle to be used by the application in the CMSCertificationResponse function call to identify the results of this request, as multiple requests may be outstanding at any time.

responseTime (CMS -> Security Application)

The estimated number of seconds until the response will be ready. Security Applications may use this estimate to determine when to perform the corresponding CMSCertificationResponse operation.

2.3.1.2 Returned Values

CMS_ContextNotRecognized

CMS_OK

CMS_UnexpectedNullPointer

2.3.2 CMSCertificationResponse

CMSCertificationResponse is used by a Registration Authority to obtain the Certification Authority response to a CMSCertificationRequest call. The response contains the authentication information that must be supplied to the end-user in order to authenticate itself to the CA in the CMSRequestMyCertificate call.

2.3.2.1 Parameters

context

See Section 2.2.2.1 above.

requestHandle (Security Application -> CMS)

See Section 2.3.1.1, above.

userReference (CMS -> Security Application)

The user reference to be used by the end-entity in its CMSRequestMyCertificate call.

authenticationToken (CMS -> Security Application)

The authentication token to be used by the end-entity in its CMSRequestMyCertificate call.

2.3.2.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidIndex

CMS_OK

CMS_UnexpectedNullPointer

2.3.3 CMSRevokeACertificate

CMSRevokeACertificate is used by a Registration Authority to revoke an end-user certificate. **DECIDE WHETHER AN ACKNOWLEDGE CALL IS REQUIRED**.

2.3.3.1 Parameters

context

See Section 2.2.2.1 above.

certificateAttributes (Security Application -> CMS)

A list of attributes that define the certificate(s) that are to be revoked..

2.3.3.2 Returned Values

CMS_ContextNotRecognized

CMS_OK

CMS_UnexpectedNullPointer

Need to add calls to support the approval by the RA of key recovery

2.4 End-user Registration Functions

These functions provide facilities whereby an end-user can register with an RA.

2.4.1 CMSRequestRegistrationForm

CMSRequestRegistrationForm requests the Registration Authority to supply a blank form for user registration.

2.4.1.1 Parameters

context

See Section 2.2.2.1 above.

responseTime

See Section 2.3.1.1 above.

2.4.1.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_OK

CMS_Timeout

CMS_UnexpectedNullPointer

2.4.2 CMSRetrieveRegistrationForm

CMSRetrieveRegistrationForm returns the blank registration form. It assumes that multiple requests to registration authorities will not be interleaved.

2.4.2.1 Parameters

context

See Section 2.2.2.1 above.

registrationForm (CMS -> Security Application)

A list of field titles for the registration form.

2.4.2.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_OK

CMS_ResponseNotReady

CMS_Timeout

CMS_UnexpectedNullPointer

2.4.3 CMSRegistrationRequest

CMSRegistrationRequest supplies the completed registration form. It assumes that multiple requests to registration authorities will not be interleaved.

2.4.3.1 Parameters

context

See Section 2.2.2.1 above.

registrationForm (Security Application -> CMS)

A list of field values for the registration form.

responseTime

See Section 2.3.1.1 above.

2.4.3.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_OK

CMS_ResponseNotReady

CMS_Timeout

CMS_UnexpectedNullPointer

2.4.4 CMSRegistrationResponse

CMSRegistrationResponse obtains the authentication information from a successful registration attempt for use in the CMSCertificationRequest call.

2.4.4.1 Parameters

context

See Section 2.2.2.1 above.

userReference (CMS -> Security Application)

The user reference by which the CA will recognize the end-user in its request for certification..

authenticationToken (CMS -> Security Application)

The authentication information which will be used to protect the authenticity, integrity and (potentially) the confidentiality of information exhcnaged by the end-user and CA in the certification process.

2.4.4.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_OK

CMS_ResponseNotReady

CMS_Timeout

CMS_UnexpectedNullPointer

2.5 Certification Functions

These functions provide facilities whereby an end-entity can obtain an initial certificate.

2.5.1 CMSRequestMyCertificate

The CMSRequestMyCertificate function is used to request a certificate, and (optionally) post it to the directory. The CMS performs no checks on user supplied keys. The operation of obtaining certificates must be completed by a call to CMSRetrieveMyCertificate.

2.5.1.1 Parameters

context

See Section 2.2.2.1 above.

userReference

See Section 2.4.4.1 above.

authenticationToken

See Section 2.4.4.1 above.

certificateAttributes (Security Application -> CMS)

A list of attributes requested for the certificate. These may be over-ridden by the attributes specified by the RA (see Section 2.3.1.1, above) or by the CA.

requestHandle

See Section 2.3.1.1 above.

responseTime

See Section 2.3.1.1 above.

2.5.1.2 Returned Values

CMS_ContextNotRecognized

CMS_IncorrectAuthenticationToken

CMS_OK

CMS_RequiredDataMissing

CMS_UnexpectedNullPointer

CMS_UserReferenceNotRecognized

CMS_CryptoLibraryNotAvailable

2.5.2 CMSRetrieveMyCertificate

CMSRetrieveMyCertificate retrieves the certificate requested by the CMSRequestMyCertificate call.

2.5.2.1 Parameters

context

See Section 2.2.2.1 above.

requestHandle (Security Application -> CMS)

The certificate handle returned by the CMSRequestMyCertificate call. See Section 2.5.1.1 above.

keyPair (CMS -> Security Application)

The certificate (path) and its component public and private keys requested by the CMSRequestMyCertificate function.

cATime

The time according to the CA. In the case of on-line commtion with the CA, this may be used to calculate the offset between the time according to the CA and that according to the user's platform.

cACertificate

An authentic CA verification certificate that can be trusted by the end-entity for use in validating certificate chains.

2.5.2.2 Returned Values

CMS_CannotAcceptUserGeneratedPrivateKey

CMS_CannotConnect

CMS_CannotGeneratePublicKey

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_IncorrectAuthenticationToken

CMS_OK

CMS_RequiredDataMissing

CMS_ResponseNotReady

CMS_SecurityProtocolFailure

CMS_Timeout

CMS_UnexpectedNullPointer

CMS_UserReferenceNotRecognized

CMS_CryptoLibraryNotAvailable

2.6 Key Update Functions

These functions provide facilities for updating a user's keys and certificates.

2.6.1 CMSKeyStatus

CMSKeyStatus returns the status of the public key contained in the certificate supplied by the Security Application. The status relates to the time validity and revocation status of the certificate and its corresponding private key. This function call may be used by the Security Application in deciding whether or not to request a key update.

2.6.1.1 Parameters

certificate (Security Application -> CMS)

The public key certificate for which the status is requested.

2.6.1.2 Returned Values

See Annex A.

2.6.2 CMSRequestKeyUpdate

The CMSRequestKeyUpdate function is used to request a new key and certificate, optionally replacing the corresponding repository entry. This function can be used by Security Applications when they have detected that their certificate must be replaced, possibly by making a call to CMSKeyStatus.

2.6.2.1 Parameters

context

See Section 2.2.2.1 above.

authenticationToken (Security Application -> CMS)

The authenticationToken supplied by the CMS in response to the most recent CMSRetrieveMyCertificate, CMSRetrieveRecoveredKeys or CMSRetrieveUpdatedKey call. If the Cryptoki library was selected in the CMSLogin function call, then this shall contain the Cryptoki PIN.

certificateAttributes (Security Application -> CMS)

A list of attributes requested for the new certificate. It must contain (as a minimum) a certificate attribute whose value is the certificate for which an update is requested.

requestHandle

See Section 2.3.1.1 above.

responseTime

See Section 2.3.1.1 above.

2.6.2.2 Returned Values

CMS_CannotAcceptUserGeneratedPrivateKey

CMS_CannotConnect

CMS_CannotGeneratePublicKey

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_OK

CMS_RequiredDataMissing

CMS_SecurityProtocolFailure

CMS_Timeout

CMS_UnexpectedNullPointer

CMS_UserUniqueNameNotRecognized

2.6.3 CMSRetrieveUpdatedKey

The CMSRetrieveUpdatedKey function is used to retrieve a new certificate, optionally replacing the existing entry in the directory. This function can be used by Security Applications following a call to CMSRequestKeyUpdate.

2.6.3.1 Parameters

context

See Section 2.2.2.1 above.

authenticationToken (Security Application -> CMS)

A new authentication token for use in subsequent update exchanges.

requestHandle (Security Application -> CMS)

See Section 2.3.1.1 above.

keyPair

The certificate (path) and its component public and private components of the keys for which a new certificate was requested.

cATime

See Section 2.5.2.1 above.

cACertificate

See Section 2.5.2.1 above.

2.6.3.2 Returned Values

CMS_CannotAcceptUserGeneratedPrivateKey

CMS_CannotConnect

CMS_CannotGeneratePublicKey

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_OK

CMS_RequiredDataMissing

CMS_ResponseNotReady

CMS_SecurityProtocolFailure

CMS_Timeout

CMS_UnexpectedNullPointer

CMS_UserUniqueNameNotRecognized

2.7 Key Recovery Functions

These functions provide facilities for recovering private keys from a Key Recovery Centre in the event that a user's record of its private key becomes lost or corrupted.

2.7.1 CMSRequestKeyRecovery

The CMSRequestKeyRecovery function is used to request the recovery of a history of keys from the KRC.

2.7.1.1 Parameters

context

See Section 2.2.2.1 above.

userReference (Security Application -> CMS)

The identifier by which the KRC identifies the user. This information is obtained by 'out-of-band' means from the KRC. The semantics of the user identifier are opaque to the Security Application.

authenticationToken (Security Application -> CMS)

A secret authentication token which is shared between the KRC and the user, and which can be used for the purpose of authenticating the user and/or confidentiality protecting private keying material exchanged between the CMS Client and the KRC. This information is obtained by 'out-of-band' means from the KRC. The semantics of the authenticationToken are opaque to the Security Application.

keyFlow (Security Application -> CMS)

Indicates whether or not the Security Application requests the CMS to generate the private key and whether or not it requests the CMS to post the certificate to the directory. If the security policy enforced by the CMS conflicts with the request, then an error code will be returned. See Section 2.1.1, above.

certificateAttributes (Security Application -> CMS)

The list of attributes describing the certificate for which recovery is requested. This should include a certificate or keyUsage attribute.

requestHandle (CMS -> Security Application)

The handle to be used by the application in the CMSRetrieveRecoveredKeys function call to obtain the results of this request.

responseTime (CMS -> Security Application)

The estimated number of seconds until the response will be ready. Security Applications may use this estimate to determine when to perform a CMSRetrieveRecoveredKeys operation to obtain the key history requested by this call.

2.7.1.2 Returned Values

CMS_ContextNotRecognized

CMS_IncorrectAuthenticationToken

CMS_OK

CMS_RequiredDataMissing

CMS_ResponseNotReady

CMS_UnexpectedNullPointer

CMS_CryptoLibraryNotAvailable

2.7.2 CMSRetrieveRecoveredKeys

The CMSRetrieveRecoveredKeys call is used to retrieve keys recovered in response to the CMSRequestKeyRecovery function call for the user identified by userReference and authenticated by authenticationToken and the key identified by keyUsage and policyId. The results are placed in the RecoveredKeyList. This list contains highly sensitive data. Therefore, it should be deleted as soon as possible following its use, by making a call to CMSEraseKeyHistory.

2.7.2.1 Parameters

context

See Section 2.2.2.1 above.

userReference

See Section above.

authenticationToken

See Section 2.7.1.1 above.

requestHandle (Security Application -> CMS)

The handle provided in the CMSRequestKeyRecovery function call.

kRCTime (CMS -> Security Application)

The time at which the response was generated by the KRC. The time shall be encoded as the number of seconds that have elapsed since midnight on the 1st of January 1970 Universal Coordinated Time. In the case of on-line retrieval, this can be used by the application to calculate the offset between the KRC's time and the application platform's time.

cACertificate

See Section 2.5.2.1 above.

2.7.2.2 Returned Values

CMS_CannotAcceptUserGeneratedPrivateKey

CMS_CannotConnect

CMS_CannotGeneratePublicKey

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_IncorrectAuthenticationToken

CMS_NoKeyHistoryAvailable

CMS_OK

CMS_RequiredDataMissing

CMS_ResponseNotReady

CMS_SecurityProtocolFailure

CMS_Timeout

CMS_UnexpectedNullPointer

CMS_UserReferenceNotRecognized

CMS_CryptoLibraryNotAvailable

2.7.3 CMSNumberOfRecoveredKeys

CMSNumberOfRecoveredKeys returns the number of keys recovered by the most recent CMSRetrieveRecoveredKeys function.

2.7.3.1 Parameters

context

See Section 2.2.2.1 above.

2.7.3.2 Returned Values

<= 0 Indicates that CMSRetrieveRecoveredKeys has not been called or did not execute correctly, or (if = 0) that it executed correctly and no keys were recovered.

> 0 Indicates the number of keys recovered.

2.7.4 CMSGetRecoveredCertificate

CMSGetRecoveredCertificate returns the certificate for the given index from RecoveredKeyList obtained by the CMSRetrieveRecoveredKeys function.

2.7.4.1 Parameters

context

See Section 2.2.2.1 above.

index (Security Application -> CMS)

The index in the RecoveredKeyList of the certificate requested by the Security Application. The value shall be between 0 and one less than the number of keys in the list (returned by CMSNumberOfRecoveredKeys) .

certificate (CMS -> Security Application)

A certificate recovered from the infrastructure by a call to CMSRetrieveRecoveredKeys.

2.7.4.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidIndex

CMS_OK

CMS_UnexpectedNullPointer

2.7.5 CMSGetRecoveredKey

CMSGetRecoveredKey returns the private key for the given index from the RecoveredKeyList obtained by the CMSRetrieveRecoveredKeys function. **Consider combining this with CMSGetRecoveredCertificate to return a keyPair.**

2.7.5.1 Parameters

context

See Section 2.2.2.1 above.

index (Security Application -> CMS)

See Section 2.7.4.1 above.

privateKey (CMS -> Security Application)

The private key recovered from the infrastructure by a call to CMSRetrieveRecoveredKeys. The internal image will be erased by a call to CMSEraseKeyHistory and CMSLogout for this context.

2.7.5.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidIndex

CMS_OK

CMS_UnexpectedNullPointer

2.7.6 CMSEraseKeyHistory

CMSEraseKeyHistory erases the key history created as a result of a call to CMSRetrieveRecoveredKeys. This function should be called as soon as possible after the Security Application has finished with the data in the RecoveredKeyList.

2.7.6.1 Parameters

context

See Section 2.2.2.1 above.

2.8 End-user Self-Revocation Functions

This function enables a user to request revocation of its own certificate.

2.8.1 CMSRevokeMyCertificate

The CMSRevokeMyCertificate function is used to request the revocation of the user's own certificate. Verification that the certificate has been revoked can be obtained by calling CMSKeyStatus.

2.8.1.1 Parameters

context

See Section 2.2.2.1 above.

authenticationToken (Security Application -> CMS)

See Section 2.7.1.1 above.

certificateAttributes (Security Application -> CMS)

Certificate attributes which uniquely define the certificate for which revocation is requested.

responseTime

See Section 2.7.1.1 above.

2.8.1.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_OK

CMS_RequiredDataMissing

CMS_SecurityProtocolFailure

CMS_Timeout

CMS_UnexpectedNullPointer

2.9 Name Resolution Functions

These functions are used to resolve unique names.

2.9.1 CMSSearchDirectory

CMSSearchDirectory searches the directory in order to build a list of directory entries which satisfy the search criteria. The results of the search are placed in the SearchResultsNameList.

2.9.1.1 Parameters

context

See Section 2.2.2.1 above.

searchExpr (Security Application -> CMS)

A pointer to a buffer containing a string expression comprising attribute type mnemonic and value pairs and logical operators, *see Ref. 2,* defining the search criteria. For example, the expression "sn=Baxter" would cause the CMS to search for all entries with a surname Baxter and the expression "(&(ou=D100)(sn=Baxter))" would cause the CMS to build a name list of all the users with the surname Baxter in the organizational unit D100.

searchBase (Security Application -> CMS)

A pointer to a buffer containing a string expression defining a portion of the DIT. For example, the expression "c=US, o=ACME" would limit the search to entries of the ACME organization of the US.

attribsToReturn (Security Application -> CMS)

A pointer to a buffer containing a list of X.500 attributes whose values are to be returned. For example, "gn\tsn\tou" (where \t represents the tab character) would return the given name, surname and organizational unit (a department name for example) of the items in the name list. The value "certificate " would cause the user certificates to be returned.

searchDepth (Security Application -> CMS)

Specifies the extent of the search in the X.500 directory.

> **SD_ObjectSearchDepth** The search is performed on only the single item specified by the search base.
>
> **SD_OneLevelSearchDepth** The search includes only the search base and one level below.

SD_SubtreeSearchDepth The search includes the search base and all levels under it down to the leaf level.

2.9.1.2 Returned Values

CMS_AttributesNotPresent

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_EntryNotFound

CMS_ImproperSearchBase

CMS_ImproperSearchExpr

CMS_OK

CMS_SearchSizeLimit

CMS_Timeout

CMS_UnexpectedNullPointer

CMS_UnrecognizedAttributes

2.9.2 CMSNumberOfNames

CMSNumberOfNames returns the number of names in a SearchResultsList.

2.9.2.1 Parameters

context

See Section 2.2.2.1 above.

2.9.2.2 Returned Values

< 0 Function could not complete.

>= 0 Number of names in the list.

2.9.3 CMSGetUniqueName

CMSGetUniqueName returns the unique name associated with an entry in the SearchResultsList.

2.9.3.1 Parameters

context

See Section 2.2.2.1 above.

nameIndex (Security Application -> CMS)

The index of the entry within the list. The index must be between 0 and one less than the number of names in the list.

uniqueName (CMS -> Security Application)

The unique name of the entry in the list.

2.9.3.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidIndex

CMS_OK

CMS_StringTruncated

CMS_UnexpectedNullPointer

2.9.4 CMSNumberOfAttributes

CMSNumberOfAttributes returns the number of attributes associated with a given name in the SearchResultsList.

2.9.4.1 Parameters

context

See Section 2.2.2.1 above.

nameIndex (Security Application -> CMS)

See Section 2.9.3.1 above.

2.9.4.2 Returned Values

< 0 Failed to complete.

2.9.5 CMSGetAttribute

CMSGetAttribute returns the mnemonic of an attribute stored within the SearchResultsList

2.9.5.1 Parameters

context

See Section 2.2.2.1 above.

nameIndex (Security Application -> CMS)

See Section 2.9.3.1 above.

attributeIndex (Security Application -> CMS)

The index of the attribute for the selected name. The index must be between 0 and one less than the number of attributes in the list. The number of attributes can be obtained by a call to CMSNumberOfAttributes. See Section 2.9.4.1, above.

attribute (CMS -> Security Application)

The mnemonic of the attribute.

2.9.5.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidAttributeIndex

CMS_InvalidListId

CMS_InvalidNameIndex

CMS_OK

CMS_StringTruncated

CMS_UnexpectedNullPointer

2.9.6 CMSNumberOfAttributeValues

CMSNumberOfAttributeValues returns the number of values associated with an attribute in the UserName List.

2.9.6.1 Parameters

context

See Section 2.2.2.1 above.

nameIndex (Security Application -> CMS)

See Section 2.9.5.1, above.

attributeIndex (Security Application -> CMS)

See Section 2.9.5.1, above.

2.9.6.2 Returned Values

< 0 Failed to complete.

2.9.7 CMSGetAttributeValue

CMSGetAttributeValue returns the value of an attribute stored within the SearchResultsList

2.9.7.1 Parameters

context

See Section 2.2.2.1 above.

nameIndex (Security Application -> CMS)

See Section 2.9.3.1 above.

attributeIndex (Security Application -> CMS)

See Section 2.9.5.1, above.

valueIndex (Security Application -> CMS)

The index of the value for the selected attribute. The index must be between 0 and one less than the number of attribute values in the list. The number of values can be obtained by a call to CMSNumberOfAttributeValues.

attribute (CMS -> Security Application)

The value of the attribute.

2.9.7.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidAttributeIndex

CMS_InvalidNameIndex

CMS_OK

CMS_StringTruncated

CMS_UnexpectedNullPointer

2.9.8 CMSResetNameList

CSMResetNameList removes all names from the SearchResultsList.

2.9.8.1 Parameters

context

See Section 2.2.2.1 above.

2.9.8.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidListId

CMS_OK

2.10 Certificate Verification Functions

These functions provide facilities for verifying another end-entity's certificate. In order to validate a certificate, a valid chain of certificates must be found linking the certificate which is to be validated to a trusted CA certificate. A suitable trusted CA certificate will have been supplied in the most recent call to CMSRetrieveMyCertificate, CMSRetrieveUpdatedKey or CMSRetrieveRecoveredKey. In the general case, this may involve the verification of some intermediate certificates. The security application may need to perform additional checks on these intermediate certificates, or it may need to record them for adjudication purposes or for diagnosing the reason for the failure of a chain validation. In the simpler case, it may be sufficient simply to indicate whether a valid certificate chain exists or not, using the checks performed by the CMS implementation. In the latter case, the CMSValidateCertificate function is used, with the parameter 'firstLink' set FALSE. Then no subsequent calls to CMSValidateNextLink are required.

In the former case, the CMSValidateCertificate function is used, with the parameter 'firstLink' set TRUE. Subsequent calls to CMSValidateNextLink are then required to complete the chain validation. This gives the security application the opportunity to record the certificates that form the certificate validation chain. This may be required in order to provide the user with diagnostic information in the event that a certificate fails to verify. In addition, if the security application wishes to apply additional checks, perhaps involving non-standard certificate extensions, then it is able to do so.

The certificate which is to be verified is specified by certificateAttributes. If the security application has the certificate then it is supplied in the 'certificate' attribute. If it does not, then the Unique Name is supplied in the 'subject' attribute or an alternative subject name is supplied in the subjectAltName attribute. The implementation will then retrieve the certificate from the SearchResultsList or the repository.

Other attributes may be supplied, in which case, it will be verified that all certificates on the chain (including the subject's) are consistent with these attributes. Upon completion, all attributes in the subject's certificate will be returned in this parameter.

2.10.1 CMSValidateCertificate

Checks that the certificate identified by certificateAttributes is valid. If the certificate attributes list include a certificate, then this will be verified. If it does not, then it must contain a Unique Name and the corresponding certificate will be obtained, either from the SearchResultsList or from the repository, for verification.

2.10.1.1 Parameters

context

See Section 2.2.2.1 above.

certificateAttributes (Security Application <-> CMS)

The list of certificate attributes which are to be verified against the contents of the certificate. Upon return, it contains the list of attributes of the verified certificate. By including reasonCodes, the security application can indicate which revocation reasons should be considered cause to reject a certificate.

cACertificate (Security Application -> CMS)

A trusted CA certificate. This forms the end of the certificate chain to be validated.

useCMSTime (Security Application -> CMS)

Indicates that the time according to the CMS shall be used to verify certificates.

userTime (Security Application -> CMS)

If useCMSTime is FALSE, then this parameter shall be used to supply the time according to the Security Application. The time shall be encoded as described in 2.7.2.1.

firstLink (Security Application -> CMS)

Indicates whether the entire certificate chain is to be validated, or just the first link in the chain. The default value is FALSE. If TRUE, then this call may be followed by one or more calls to CMSValidateNextLink.

allowPolicyMapping (Security Application -> CMS)

Indicates that policy mapping may be used in forming the certificate verification path.

revalidateAt (CMS -> Security Application)

The time at which the certificate should be revalidated. **Need to think about this some more. Two cases must be considered. 1. The validAt parameter, specified by the application, is prior to the thisUpdate field of the current CRL. Then the validity of the certificate can be determined with certainty. 2. The validAt parameter is between the current CRL's thisUpdate and nextUpdate times. In this case, the validity is conditional upon revalidating the certificate after the current CRL's nextUpdate time. The validAt parameter should never be greater than the present time.**

2.10.1.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_InvalidCertificateSyntax

CMS_OK

CMS_RequiredDataMissing

CMS_Timeout

CMS_UnexpectedNullPointer

2.10.2 CMSValidateNextLink

Returns the next valid certificate in the certificate chain. There may be multiple consecutive calls to this function: one for each certificate in the chain. Such a sequence must be preceded by a call to CMSValidateCertificate. This function should be called repeatedly until the value CMS_OK is returned. Upon return, the certificateAttributes parameter contains attributes of the next certificate in the chain. The security application may perform additional checks on non-standard extensions and indicate whether or not the certificate is acceptable by means of the 'continue' parameter. If the CMS implementation runs into a dead-end, then it must back-track to an earlier acceptable certificate which has at least one unexplored branch. When this happens, it returns a positive integer value in the 'backtrack' parameter. An application should respond by removing that number of good certificates from its list, in a last-in-first-out manner.

2.10.2.1 Parameters

context

See Section 2.2.2.1 above.

continue (Security Application -> CMS)

Indicates whether the last certificate was valid or not according to additional checks that may be performed by the security application. Default TRUE.

certificate (CMS -> Security Application)

The next certificate in the chain.

backTrack (CMS -> Security Application)

Indicates the number of certificates in the path which should be discarded. If the value is non-zero, then the implementation has explored a branch in the CA network which is a dead-end.

2.10.2.2 Returned Values

CMS_CannotConnect

CMS_ContextNotRecognized

CMS_DirectoryAccessDenied

CMS_InvalidCertificateSyntax

CMS_IncompleteChain

CMS_OK

CMS_RequiredDataMissing

CMS_Timeout

CMS_UnexpectedNullPointer

2.10.3 CMSGetCertificateAttribute

This function returns the value of an attribute of the certificate validated by the most recent call to CMSValidateCertificate. The type of the required attribute is referenced by its attribute type object identifier. Some attributes may have multiple values. In this case, the valueIndex parameter is used to select one value.

2.10.3.1 Parameters

context

See Section 2.2.2.1 above.

certificate (Security Application -> CMS)

The certificate from which an attribute is to be extracted.

attributeType (Security Application -> CMS)

See Section 2.1 above.

valueIndex (Security Application -> CMS)

Indicates the attribute value to be returned in the case where the attribute has multiple values (for instance certificatePolicy). NULL indicates the first value.

attributeValue (CMS -> Security Application)

The value of the attribute.

lastValue (CMS -> Security Application)

Indicates that the requested value is the last in the list of certificate attribute values.

2.10.3.2 Returned Values

CMS_ContextNotRecognized

CMS_InvalidAttributeType

CMS_InvalidValueIndex

CMS_OK

2.11 Utility Functions

This section contains a set of utility functions.

2.11.1 CMSReleaseBuffer

This function releases memory which has been allocated by the implementation.

2.11.1.1 Parameters

buffer (Security Application -> CMS)

The buffer for which the allocated memory is to be released.

2.11.1.2 Returned Values

CMS_OK

CMS_UnexpectedNullPointer

2.11.2 CMSReleaseName

This function releases memory which has been allocated by the implementation.

2.11.2.1 Parameters

name (Security Application -> CMS)

The name for which the allocated memory is to be released.

2.11.2.2 Returned Values

CMS_OK

CMS_UnexpectedNullPointer

2.11.3 CMSReleaseKey

This function releases memory which has been allocated by the implementation.

2.11.3.1 Parameters

keyPair (Security Application -> CMS)

The key pair for which the allocated memory is to be released.

2.11.3.2 Returned Values

CMS_OK

CMS_UnexpectedNullPointer

2.11.4 CMSGetLogString

Returns a textual explanation of a CMSLog code. This function is like CMSLogToString(), except that the textual explanation is returned through the function's second parameter rather than through its return value.

2.11.4.1 Parameters

log (Security Application -> CMS)

The CMSLog code to be described.

explanation (CMS -> Security Application)

A textual description of the CMSLog code.

2.11.4.2 Returned Values

CMS_LogValueNotRecognized

CMS_UnexpectedNullPointer

2.11.5 CMSLogToString

Returns a textual explanation of a CMSLog code.

2.11.5.1 Parameters

log (Security Application -> CMS)

The CMSLog code to be translated.

2.11.5.2 Returned Values

A string describing the CMSLog code.

2.11.6 CMSQueryLogWarning

Indicates whether a CMSLog is an error or a warning.

2.11.6.1 Parameters

log (Security Application -> CMS)

2.11.6.2 Returned Values

TRUE The CMSLog value is a warning.

FALSE The CMSLog value is an error.

2.11.7 CMSQueryVersionNumber

Returns a string indicating which version of the CMS is running.

2.11.8 Name Conversion Functions

The following functions, as defined in refs 7 and 8, shall be supported.

gss_compare_name

gss_display_name

gss_export_name

gss_import_name

gss_inquire_name_for_mech

gss_release_name

3. Status Codes

The status codes are values of the enum type CMSLog. **ADD GSS STATUS CODES**

3.1 Error Codes

CMS_AttributesNotPresent One or more of the requested attributes is not present in the directory.

CMS_CannotAcceptUserGeneratedPrivateKey The CMS Security Policy does not permit users to generate key pairs of the requested type.

CMS_CannotConnect The CMS could not establish a connection to the CA or to the directory.

CMS_CannotGeneratePublicKey The CMS Security Policy does not permit the CA to generate the user's key pair of the requested type.

CMS_CannotGetCRL

CMS_ContextNotRecognized The context handle was not recognized.

CMS_CryptoLibraryNotAvailable The library of crypto functions could not be located.

CMS_CryptoLibraryNotSupported The selected library of crypto functions is not supported by the CMS.

CMS_CryptoLibraryPINIncorrect The PIN supplied for the crypto library has been rejected.

CMS_DirectoryAccessDenied Access to the directory was denied.

CMS_EntryNotFound

CMS_FileError

CMS_FormattingError

CMS_IncompleteChain The certificate could not be verified, because no complete certificate chain could be found.

CMS_IncorrectAuthenticationToken The supplied authentication token is incorrect.

CMS_InvalidAttributeIndex The AttributeIndex is out of range.

CMS_InvalidAttributeMnemonic The name is not a valid attribute mnemonic.

CMS_InvalidCertificateSyntax The syntax of the certificate is not standard compliant.

CMS_InvalidFieldName

CMS_InvalidListId The ListId is not recognized.

CMS_InvalidMode

CMS_InvalidNameIndex The NameIndex is out of range.

CMS_InvalidParm

CMS_InvalidPathEntryIndex The index of the requested path entry is not valid.

CMS_InvalidSearchBase The search base is not properly formed.

CMS_InvalidSearchExpr The search expression is not properly formed.

CMS_InvalidValueIndex The ValueIndex is out of range.

CMS_ManagerClientTimeMismatch

CMS_MaximumNumberOfOpenContextsExceeded The maximum number of open contexts has been exceeded.

CMS_MemoryError

CMS_NoKeyHistoryAvailable The CA is unable to provide a key history.

CMS_RequiredDataMissing Required data is missing from the function call.

CMS_ResponseNotReady The CMS has not yet responded to a request for a certificate, key update or key recovery.

CMS_SecurityProtocolFailure The security protocol between the CMS Client and the CA failed.

CMS_StateError

CMS_Timeout A directory access timed out.

CMS_UnexpectedNullPointer A pointer was found to be null.

CMS_UnknownError

CMS_Unsupported

CMS_UserReferenceNotRecognized The CA does not recognize the supplied userReference.

CMS_UserUniqueNameNotRecognized The CA does not recognize the supplied User UniqueName.

CMS_VersionNumberNotSupported The Version number is not supported by the CMS.

3.2 Warning Codes

CMS_SearchSizeLimit The data retrieved from the directory is incomplete.

CMS_StringTruncated The retrieved data exceeds the buffer size.

3.3 Normal Operation Codes

CMS_OK The function completed successfully.

4. Certificate Attribute Object Identifiers

Certificate attributes are used in a number of different ways. They may be specified by the end-user at the time of requesting a certificate. They may also be specified by an end-user as acceptance criteria when verifying a certificate. Finally, they represent the contents of a verified certificate.

There is broad range of sophistication amongst client applications. Some simply require to be able to supply a DN and receive a verified public key in return, these do not want to be burdened by complex options. At the other end of the scale, there are applications which must impose complex constraints related to policy.

The certificate attributes, in all of these cases, will be dealt with in the same way: as a list of attribute/value pairs. The attributes are expressed as a CMSBinaryData type whose contents is the attribute OID (as defined below) in 'stop-separated' format, with each character ASCII-encoded. The attribute values are encoded as CMSBinaryData types, whose contents is a 'C' type equivalent to the ASN.1 type of the corresponding certificate field. This rule only applies to certificate fields and extension fields which are 'independent'. In other words, if an extension contains fields which must be interpretted in conjunction with other fields, then the whole extension is passed in the list. The structure of such extensions is defined in terms of a 'C' structure, below, but only the extension is assigned an OID (not its individual fields), and the whole extension is passed in the list.

Object Identifier Usage

certificateAttribute	ID	::=	TBD
id-ca	ID	::=	certificateAttribute

Certificate Attributes

certificate

certificate	OBJECT IDENTIFIER ::= {id-at 36}
version	OBJECT IDENTIFIER ::= {id-ca 1}
serialNumber	OBJECT IDENTIFIER ::= {id-ca 2}
signature	OBJECT IDENTIFIER ::= {id-ca 3}
issuer	OBJECT IDENTIFIER ::= {id-ca 4}
validityNotBefore	OBJECT IDENTIFIER ::= {id-ca 5}
validityNotAfter	OBJECT IDENTIFIER ::= {id-ca 6}
subject	OBJECT IDENTIFIER ::= {id-ca 7}
subjectPublicKeyInfoAlgorithmAlgorithm	OBJECT IDENTIFIER ::= {id-ca 8}
subjectPublicKeyInfoAlgorithmParameters	OBJECT IDENTIFIER ::= {id-ca 9}
subjectPublicKeyInfoAlgorithmSubjectPublicKey	OBJECT IDENTIFIER ::= {id-ca 10}

issuerUniqueIdentifier OBJECT IDENTIFIER ::= {id-ca 11}
subjectUniqueIdentifier OBJECT IDENTIFIER ::= {id-ca 12}

authorityKeyIdentifier

authorityKeyIdentifier OBJECT IDENTIFIER ::= {id-ca 13}
authorityCertIssuer OBJECT IDENTIFIER ::= {id-ca 14}
authorityCertSerialNumber OBJECT IDENTIFIER ::= {id-ca 15}

subjectKeyIdentifier

subjectKeyIdentifier OBJECT IDENTIFIER ::= {id-ca 16}
subjectCertIssuer OBJECT IDENTIFIER ::= {id-ca 17}
subjectCertSerialNumber OBJECT IDENTIFIER ::= {id-ca 18}

keyUsage

privateKeyUsagePeriodNotBefore OBJECT IDENTIFIER ::= {id-ca 19}
privateKeyUsagePeriodNotAfter OBJECT IDENTIFIER ::= {id-ca 20}

certificatePolicies

certificatePolicies OBJECT IDENTIFIER ::= {id-ca 21}

policyMappings

policyMappings OBJECT IDENTIFIER ::= {id-ca 22}

subjectAltName

subjectAltNameOtherName OBJECT IDENTIFIER ::= {id-ca 23}
subjectAltNameRfc822Name OBJECT IDENTIFIER ::= {id-ca 24}
subjectAltNameDNSName OBJECT IDENTIFIER ::= {id-ca 25}
subjectAltNameX400Name OBJECT IDENTIFIER ::= {id-ca 26}
subjectAltNameDirectoryName OBJECT IDENTIFIER ::= {id-ca 27}

subjectAltNameEdiPartyName OBJECT IDENTIFIER ::= {id-ca 28}
subjectAltNameUniformResourceIdentifier OBJECT IDENTIFIER ::= {id-ca 29}
subjectAltNameIPAddress OBJECT IDENTIFIER ::= {id-ca 30}
subjectAltNameRegisteredId OBJECT IDENTIFIER ::= {id-ca 31}

issuerAltName

issuerAltNameOtherName OBJECT IDENTIFIER ::= {id-ca 32}
issuerAltNameRfc822Name OBJECT IDENTIFIER ::= {id-ca 33}
issuerAltNameDNSName OBJECT IDENTIFIER ::= {id-ca 34}
issuerAltNameX400Name OBJECT IDENTIFIER ::= {id-ca 35}
issuerAltNameDirectoryName OBJECT IDENTIFIER ::= {id-ca 36}
issuerAltNameEdiPartyName OBJECT IDENTIFIER ::= {id-ca 37}
issuerAltNameUniformResourceIdentifier OBJECT IDENTIFIER ::= {id-ca 38}
issuerAltNameIPAddress OBJECT IDENTIFIER ::= {id-ca 39}
issuerAltNameRegisteredId OBJECT IDENTIFIER ::= {id-ca 40}

subjectDirectoryAttribute

subjectDirectoryAttribute OBJECT IDENTIFIER ::= {id-ca 41}

basicConstraints

NOT SUPPORTED - if this extension occurs in an end-user certificate, then its contents is entirely predictable. If this extension occurs in a CA certificate, then all associated processing will be performed by the implementation.

nameConstraints

NOT SUPPORTED - if this extension occurs in an end-user certificate, then its contents is entirely predictable. If this extension occurs in a CA certificate, then all associated processing will be performed by the implementation.

policyConstraints

policyConstraints OBJECT IDENTIFIER ::= {id-ca 42}

reasonCode

This certificate attribute may be used by certificate users to indicate which revocation reasons are grounds for rejecting a certificate.

reasonCodeUnspecified OBJECT IDENTIFIER ::= {id-ca 43}
reasonCodeKeyCompromise OBJECT IDENTIFIER ::= {id-ca 44}
reasonCodeCACompromise OBJECT IDENTIFIER ::= {id-ca 45}
reasonCodeAffiliationChanged OBJECT IDENTIFIER ::= {id-ca 46}
reasonCodeSuperseded OBJECT IDENTIFIER ::= {id-ca 47}
reasonCodeCessationOfOperation OBJECT IDENTIFIER ::= {id-ca 48}
reasonCodeCertificateHold OBJECT IDENTIFIER ::= {id-ca 49}
reasonCodeRemoveFromCRL OBJECT IDENTIFIER ::= {id-ca 50}

holdInstructionCode

holdInstructionCode OBJECT IDENTIFIER ::= {id-ca 51}

invalidityDate

This certificate attribute is returned to certificate user applications to indicate the time at which a revoked certificate was known to be invalid.

invalidityDate OBJECT IDENTIFIER ::= {id-ca 52}

crlDistributionPoints

NOT SUPPORTED - either this does not occur in end-user certificates, or its value is predictable

issuingDistributionPoints

NOT SUPPORTED - either this does not occur in end-user certificates, or its value is predictable

certificateIssuer

NOT SUPPORTED - This does not occur in end-user certificates, or its value is predictable

deltaCRLIndicator

NOTE: A method is needed to force the implementation to use on-line CRLs.

unspecified non-critical extension

unspecifiedNonCriticalExtension OBJECT IDENTIFIER ::= {id-ca 53}

unspecified critical extension

unspecifiedCriticalExtension OBJECT IDENTIFIER ::= {id-ca 54}

validTime

This certificate attribute is supplied by the certificate user applciation to indicate the time at which the validity of the certificate is required.

validTime OBJECT IDENTIFIER ::= {id-ca 55}

5. References

1. PKCS#1, RSA Encryption Standard, v1.5, RSA Laboratories, Nov 93.

2. RFC 1558, A string representation of LDAP search filters, T Howes, Feb 1995.

3. The Directory: Authentication Framework, ITU-T Recommendation X.509, 1993.

4. Draft Amendment 1 to ITU-T Recommendation X.509 (1993 E), Aug 1995.

5. RFC 1779, A string representation of Dictinguished Names, S Kille, Mar 1995.

6. RFC 1778, The string representation of standard attribute syntaxes, T Howes, S Kille, W Yeong, C Robbins, Mar 1995.

7. RFC 1508, Generic Security Service Application Program Interface, J Linn, Sep 1993.

8. RFC 1509, Generic Security Service API: C-bindings, J Wray, Sep 1993.

9. A text that describes CA networks.

ANNEX A - SAMPLE SECURITY APPLICATION PSEUDO-CODE

ANNEX B - C LANGUAGE BINDINGS

```
/* cmsbdefs.h - header file for the CMS API basic definitions. */

ifndef CMSBDEFS_H
#define CMSBDEFS_H

typedef int                 BOOL;
typedef short int           INT16;
typedef unsigned short int  UINT16;
typedef long                INT32;
typedef unsigned long       UINT32;
typedef unsigned char       BYTE;
typedef unsigned int        UINT;
typedef int                 INT;
typedef INT32               CMSLog;

#ifndef  TRUE
#define  TRUE 1
#define FALSE 0
#endif

#define CMS_MaxStringSize  32767

typedef enum {
        CL_Built-in,            /* Use the CMS internal library of
                                   crypto functions */
        CL_Cryptoki             /* Use the Cryptoki library of crypto
                                   functions */
        } CryptoLibrary;

typedef enum {
        KS_Valid,               /* The certificate is valid
        KS_NotYetValid,         /* The notBefore parameter of the
                                   certificate's validity is later than the
                                   current time */
        KS_KeyUpdateRequired,   /* The private key corresponding to
                                   the certificate is still valid, however,
                                   according to the CMS's security
                                   policy, the Security Application
                                   should attempt to update the key */
        KS_PrivateKeyExpired,   /* The current time is later than the
                                   notAfter parameter of the
                                   privateKeyValidity in the
```

```
                                        certificate's keyAttribute extension
                                        */
        KS_PublicKeyExpired             /* The current time is later than the
                                        notAfter parameter of the
                                        certificate's validity */
        KS_Revoked                      /* The certificate has been revoked
                                        */
        } KeyStatus;

typedef enum {
        SD_ObjectSearchDepth,           /* Search for the specified entry only
                                        */
        SD_OneLevelSearchDepth,         /* Search in the sub-tree including
                                        one-level below the specified entry
                                        */
        SD_SubtreeSearchDepth           /* Search in the entire sub-tree below
                                        the specified entry */
        } SearchDepth;

typedef enum {
        VFR_OK,                         /* The certificate verified
                                        successfully */
        VFR_CertificateExpired,         /* The certificates has expired */
        VFR_CertificateNotFound,        /* It was not possible to retrieve a
                                        certificate for the issuer of the
                                        certificate */
        VFR_CertificateNotYetValid,     /* The certificates is not yet valid */
        VFR_CertificateSigningAlgorithmNotRecognized,  /* The algorithm used to sign
                                        the certificate was not recognized */
        VFR_DirectoryAccessDenied,      /* Access to the directory was denied
                                        */
        VFR_EntryNotFound,              /* It was not possible to find a
                                        directory entry for the issuer of the
                                        certificate */
        VFR_InvalidCertificateSyntax,   /* The syntax of the certificate is not
                                        standard compliant */
        VFR_SignatureFailure,           /* The signature on the certificate
                                        failed to verify */
        VFR_UnrecognizedCriticalExtension,  /* The certificate contains a critical
                                        certificate extension which was not
                                        recognized */
        VFR_UntrustedCA,                /* A certificate chain to a trusted CA
                                        could not be found */
        VFR_CertificateRevoked,         /* The certificate has been revoked
                                        */
```

```
    VFR_CannotGetValidCRL,          /* It was not possible to obtain a
                                       valid CRL */
    VFR_Unknown                     /* A failure reason other than those
                                       listed above was encountered */
    } VerifyFailureReason;

typedef enum{
    DT_ByteString,                  /* an ASN.1 OCTET STRING */
    DT_CharString,                  /* a null terminated character string
                                       */
    DT_Integer,
    DT_Time,                        /* time_t */
    DT_BER                          /* a BER-encoded ASN.1 structure
                                       */
} CMSDataType;

typedef void* CMSContext;

typedef struct {
    uint32  size;                   /* The size (in Bytes) of the binary
                                       data */
    BYTE*  data;
    } CMSBinaryData;

typedef struct {
    BOOL keyFromCMS;                /* If TRUE, then the CMS generates
                                       the user's private key, otherwise the
                                       Security Application generates it */
    BOOL certificateToCMS;          /* If TRUE, then the certificate is
                                       posted by the CMS, otherwise the
                                       Security Application must post it */
    } KeyFlow;

typedef enum {
    UINT32          KeyLength;
    PublicKeyType   keyType;
    } PublicKeyType;

typedef struct {
    PKT_RSA = 0,
    PKT_DSA,
    PKT_DEFAULT
    } PublicKeyParameters;
```

```
typedef struct {
        CMSBinaryData       privateKey;
        CMSBinaryData       publicKey;
        PublicKeyParameters publicKeyParameters;
        CMSBinaryData       certificate;
        } KeyPair;

/* certificate attribute declarations */
CMSBinaryData       certificate;     /* the contents of the CMSBinaryData type is the
                                     /* certificate

CMSBinaryData       version;         /* the contents of the CMSBinaryData type is
                                     /* versionContents

int                 versionContents;

CMSBinaryData       serialNumber;    /* the contents of the CMSBinaryData type is
                                     /* serialNumberContents

int                 serialNumberContents;

CMSBinaryData       signature Algorithm;  /* the contents of theCMSBinaryDatatype  is
                                          /* the signature algorithm identifier, in
‘stop-
                                          /* separated’ OID format, with each
character
                                          /* ASCII encoded.

CMSBinaryData       signatureParameters;  /* the contents of the CMSBinaryDatatype is
                                          /* the DER-encoding of the parameters
                                          /* structure in the certificate.

CMSBinaryData       issuer;               /* the contents of the CMSBinaryData type
is
                                          /* issuerContents

typedef struct {
        CMSBianryData       issuerNameType;
        CMSBianryData       issuerCharSet;
        CMSBinaryData       issuerName;
        } issuerContents;

CMSBinaryData       validityNotBefore ;   /* the contents of the CMSBinaryData type
is
                                          /* validityNotBeforeContents
```

```
int             validityNotBeforeContents;  /* the number of seconds from
                                            /* midnight on the 1st of Jan 1970
                                            /* Universal Coordinated Time until
                                            /* the start of the certificate validity
                                            /* period

CMSBinaryData   validityNotAfter  ;   /* the contents of the CMSBinaryData type
is
                                      /* validityNotAfterContents

int             validityNotAfterContents;   /* the number of seconds from
                                            /* midnight on the 1st of Jan 1970
                                            /* Universal Coordinated Time until
                                            /* the end of the certificate validity
                                            /*  period

CMSBinaryData   subject:              /* the contents of the CMSBinaryData type
is
                                      /* subjectContents

typedef struct {
        CMSBianryData       subjectNameType;
        CMSBianryData       subjectCharSet;
        CMSBinaryData       subjectName;
        } subjectContents;

CMSBinaryData   subjectPublicKeyInfoAlgorithm ;     /* the contents of the
                                                    /* CMSBinaryDatatype  is
the
                                                    /* algorithm identifier of
                                                    /* the subject's algorithm, in
                                                    /* 'stop-separated' OID
                                                    /* format, with each
                                                    /* character ASCII encoded

CMSBinaryData   subjectPublicKeyInfoParameters ;    /* the contents of the
                                                    /* CMSBinaryDatatype is the
                                                    /* DER-encoding of the
                                                    /* parameters structure in the
                                                    /* subjectPublicKeyInfo

CMSBinaryData   subjectPublicKeyInfoPublicKey ;     /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* DER-encoded ASN.1
```

```
                                                    /* SEQUENCE of n and e

CMSBinaryData       issuerUniqueIdentifier ;        /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* BER-encoded ASN.1
                                                    /* UniqueIdentifier

CMSBinaryData       subjectUniqueIdentifier ;       /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* BER-encoded ASN.1
                                                    /* UniqueIdentifier

CMSBinaryData       authorityKeyIdentifier;         /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* authorityKeyIdentifier
                                                    /* Contents

int                 authorityKeyIdentifierContents;

CMSBinaryData       authorityCertIssuer;            /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* authorityCertIssuer
                                                    /* Contents

typedef struct {
        CMSBianryData       authorityNameType;
        CMSBianryData       authorityCharSet;
        CMSBinaryData       authorityName;
        } authorityCertIssuerContents;

CMSBinaryData       authorityCertSerialNumber;      /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* authorityCertIssuerSerial
                                                    /* NumberContents

int                 authorityCertIssuerSerialNumberContents;

CMSBinaryData       subjectKeyIdentifier;           /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* subjectKeyIdentifier
```

```
                                                    /* Contents

int                 subjectKeyIdentifierContents ;

CMSBinaryData       subjectCertIssuer;              /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* subjectCertIssuer
                                                    /* Contents

typedef struct {
        CMSBianryData       subjectNameType;
        CMSBianryData       subjectCharSet;
        CMSBinaryData       subjectName;
        } subjectCertIssuerContents;

CMSBinaryData       subjectCertSerialNumber;        /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* subjectCertSerialNumber
                                                    /* Contents

int                 subjectCertSerialNumberContents;

CMSBinaryData       privateKeyUsagePeriodNotBefore; /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* privateKeyNotBefore
                                                    /* Contents

int                 privateKeyNotBeforeContents;    /* the number of seconds
                                                    /* from midnight on the 1st
of
                                                    /* Jan 1970
                                                    /* Universal Coordinated
                                                    /*Time until the start of the
                                                    /* private key validity
                                                    /*  period

CMSBinaryData       privateKeyUsagePeriodNotAfter;  /* the contents of the
                                                    /* CMSBinaryData type is
the
                                                    /* privateKeyNotAfter
                                                    /* Contents
```

```
int             privateKeyNotAfterContents;     /* the number of
seconds
                                                /* from midnight on the 1st
of
                                                /* Jan 1970
                                                /* Universal Coordinated
                                                /* Time until the end of the
                                                /* private key
                                                /* validity period

CMSBinaryData   certificatePolicies;    /* the contents of theCMSBinaryDatatype  is
                                        /* one or more certificatePoliciesContents

typedef struct {
        CMSBinaryData       policyIdentifier ;
        PolicyQualifierInfo policyqualifiers;
        } certificatePoliciesContents;

typedef struct {
        CMSBinaryData       policyQualifierId;
        CMSBinaryData       qualifier;      /* one or more QualifierInfo structures
        } policyQualifierInfo;

typedef struct {
        CMSBinaryData       policyQualifierId;
        CMSBinaryData       qualifier;
        } QualifierInfo;

CMSBinaryData   policyMappings;     /* the contents of the CMSBinaryData type
is
                                    /* policyMappingsContents

typedef struct {
        CMSBinaryData       issuerPolicy;   /* 'stop-separated' OID format
        CMSBinaryData       subjectPolicy; /* 'stop-separated' OID format
        } policyMappingsContents;

CMSBinaryData   subjectAltNameOtherName ; /* the contents of the
CMSBinaryData
                                          /* type is thesubject OtherName
                                          /* Contents

typedef struct {
        CMSBinaryData       nametype;       /* 'stop-separated' OID format
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                otherName;
```

```
        } subjectOtherNameContents      /* the memory is allocated by the
                                        /* implementation, and must be released by
a
                                        /* call to
                                        /* CMSReleaseName

CMSBinaryData   subjectAltNameRfc822Name;           /* the contents of the
                                                    /* CMSBinaryData type is
                                                    /* thesubject RfcName
                                                    /* Contents
typedef struct {
        CMSBinaryData   charSet;    /* 'stop-separated' OID format
        char            rfcName;
        } subjectRfcNameContents    /* the memory is allocated by the implementation,
                                    /* and must be released by a call to
                                    /* CMSReleaseName

CMSBinaryData   subjectAltNameDNSName;  /* the contents of the
CMSBinaryData
                                        /* type is thesubject DNSName
                                        /* Contents

typedef struct {
        CMSBinaryData   charSet;    /* 'stop-separated' OID format
        char            dNSName;
        } subjectDNSNameContents    /* the memory is allocated by the implementation,
                                    /* and must be released by a call to
                                    /* CMSReleaseName

CMSBinaryData   subjectAltNameX400Name ;    /* the contents of the
CMSBinaryData
                                        /* type is thesubject X400Name
                                        /* Contents

typedef struct {
        CMSBinaryData   charSet;    /* 'stop-separated' OID format
        char            x400Name;
        } subjectX400NameContents   /* the memory is allocated by the implementation;
                                    /* and must be released by a call to
                                    /* CMSReleaseName

CMSBinaryData   subjectAltNameDirectoryName ;   /* the contents of the
                                                /* CMSBinaryData type is
                                                /* thesubject DirectoryName
                                                /* Contents
```

```
typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                directoryName;
        } subjectDirectoryNameContents      /* the memory is allocated by the
                                            /* implementation, and must be released by
a
                                            /* call to
                                            /* CMSReleaseName

CMSBinaryData       subjectAltNameEdiPartyName ;      /* the contents of the
                                                      /* CMSBinaryData type is
                                                      /* thesubject EdiName
                                                      /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                ediName;
        } subjectEdiNameContents    /* the memory is allocated by the implementation,
                                    /* and must be released by a call to
                                    /* CMSReleaseName

CMSBinaryData       subjectAltNameUniformResourceIdentifier; /* the contents of the
                                                             /* CMSBinaryData
                                                             /* type is thesubject
                                                             /* UrlName Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                urlName;
        } subjectUrlNameContents    /* the memory is allocated by the implementation,
                                    /* and must be released by a call to
                                    /* CMSReleaseName

CMSBinaryData       subjectAltNameIPAddress ;   /* the contents of the
CMSBinaryData
                                                /* type is thesubject IPName
                                                /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                iPName;
        } subjectIPNameContents     /* the memory is allocated by the
                                    /* implementation, and must be
                                    /* released by a call to CMSReleaseName

CMSBinaryData       subjectAltNameRegisteredId;       /* the contents of the
```

```
                                                    /* CMSBinaryData type is
                                                    /* the subjectRegistered
                                                    /* IdNameContents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                registeredIdName;
        } subjectRegisteredIdNameContents /* the memory is allocated by the
                                          /* implementation, and must be
                                          /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameOtherName ;  /* the contents of the
CMSBinaryData
                                              /* type is theissuer OtherName
                                              /* Contents

typedef struct {
        CMSBinaryData       nametype;       /* 'stop-separated' OID format
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                otherName;
        } issuerOtherNameContents         /* the memory is allocated by the
                                          /* implementation, and must be
                                          /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameRfc822Name;        /* the contents of the
                                                    /* CMSBinaryData type is
                                                    /* theissuer RfcName
                                                    /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                rfcName;
        } issuerRfcNameContents     /* the memory is allocated by the
                                    /* implementation, and must be
                                    /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameDNSName;   /* the contents of the
                                            /* CMSBinaryData type is
                                            /* theissuer DNSName
                                            /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                dNSName;
        } issuerDNSNameContents     /* the memory is allocated by the
                                    /* implementation, and must be
```

```
                                /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameX400Name ;   /* the contents of the
                                              /* CMSBinaryData type is
                                              /* theissuer X400Name
                                              /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                x400Name;
        } issuerX400NameContents  /* the memory is allocated by the
                                  /* implementation, and must be
                                  /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameDirectoryName ;      /* the contents of the
                                                      /* CMSBinaryData type is
                                                      /* theissuer DirectoryName
                                                      /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                directoryName;
        } issuerDirectoryNameContents     /* the memory is allocated by the
                                          /* implementation, and must be
                                          /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameEdiPartyName ;       /* the contents of the
                                                      /* CMSBinaryData type is
                                                      /* theissuer EdiName
                                                      /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                ediName;
        } issuerEdiNameContents     /* the memory is allocated by the
                                    /* implementation, and must be
                                    /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameUniformResourceIdentifier;   /* the contents of the
                                                              /* CMSBinaryData
                                                              /* type is theissuer
                                                              /* UrlName
                                                              /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
```

```
        char                urlName;
        } issuerUrlNameContents     /* the memory is allocated by the
                                    /* implementation, and must be
                                    /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameIPAddress ;    /* the contents of the
CMSBinaryData
                                                /* type is theissuer IPName
                                                /* Contents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                iPName;
        } issuerIPNameContents      /* the memory is allocated by the
                                    /* implementation, and must be
                                    /* released by a call to CMSReleaseName

CMSBinaryData       issuerAltNameRegisteredId;  /* the contents of the
                                                /* CMSBinaryData type is
                                                /* the issuerRegistered
                                                /* IdNameContents

typedef struct {
        CMSBinaryData       charSet;        /* 'stop-separated' OID format
        char                registeredIdName;
        } issuerRegisteredIdNameContents    /* the memory is allocated by the
                                            /* implementation, and must be
                                            /* released by a call to CMSReleaseName

CMSBinaryData       subjectDirectoryAttribute;  /* the contents of the
CMSBinaryData
                                                /* type is the DER-encoded subject
                                                /* DirectoryAttribute, as it is in the
                                                /* certificate

CMSBinaryData       policyConstraints;      /* the contents of the CMSBinaryData
                                            /* type is policyConstraintsContents

typedef struct {
        CMSBinaryData       PolicySet;      /* 'stop-separated' OID format
        int                 RequireExplicitPolicy;
        int                 InhibitPolicyMapping;
        } policyConstraintsContents

CMSBinaryData       reasonCodeUnspecified;      /* the contents of the
CMSBinaryData
```

```
                                        /* type is
unspecifiedReasonContents

BOOL            unspecifiedReasonContents;

CMSBinaryData   reasonCodeKeyCompromise  ;   /* the contents of the
                                            /* CMSBinaryData
                                            /* type is keyCompromise
                                            /* ReasonContents

BOOL            keyCompromiseReasonContents;

CMSBinaryData   reasonCodeCACompromise;     /* the contents of the
                                            /* CMSBinaryData
                                            /* type is cACompromise
                                            /* ReasonContents

BOOL            cACompromiseReasonContents;

CMSBinaryData   reasonCodeAffiliationChanged;   /* the contents of the
                                            /* CMSBinaryData
                                            /* type is affiliationChange
                                            /* ReasonContents

BOOL            affiliationChangeReasonContents;

CMSBinaryData   reasonCodeSuperseded;       /* the contents of the
                                            /* CMSBinaryData
                                            /* type is superseded
                                            /* ReasonContents

BOOL            supersededReasonContents;

CMSBinaryData   reasonCodeCessationOfOperation; /* the contents of the
                                            /* CMSBinaryData
                                            /* type is cessation
                                            /* ReasonContents

BOOL            cessationReasonContents;

CMSBinaryData   reasonCodeCertificateHold;  /* the contents of the
                                            /* CMSBinaryData
                                            /* type is certificateHold
                                            /* ReasonContents

BOOL            certificateHoldReasonContents;
```

```
CMSBinaryData       reasonCodeRemoveFromCRL;        /* the contents of the
                                                    /* CMSBinaryData
                                                    /* type is removeFromCRL
                                                    /* ReasonContents

BOOL                removeFromCRLReasonContents;

CMSBinaryData       holdInstructionCode;            /* the contents of
                                                    /* theCMSBinaryDatatype  is
                                                    /* the hold instruction
                                                    /* identifier, in 'stop-
                                                    /* separated' OID format,
                                                    /* with each character
                                                    /* ASCII encoded.

CMSBinaryData       invalidityDate;                 /* the contents of the
CMSBinaryData type is
                                                    /* invalidityContents

int                 invalidityContents;             /* the number of seconds from
                                                    /* midnight on the 1st of Jan 1970
                                                    /* Universal Coordinated Time until
                                                    /* the time of the certificate invalidity

CMSBinaryData       unspecifiedNonCriticalExtension; /* the contents of the
                                                    /* CMSBinaryData type is
                                                    /* the DER-encoded ASN.1
                                                    /* structure of the extension

CMSBinaryData       unspecifiedCriticalExtension;       /* the contents of the
                                                    /* CMSBinaryData type is
                                                    /* the DER-encoded ASN.1
                                                    /* structure of the extension

CMSBinaryData       validTime;      /* the contents of the CMSBinaryData type is
                                    /* validTimeContents

int                 validTimeContents;              /* the time in seconds since midnight on 1st
                                                    /* of Jan 1970 until the time at which the
                                                    /* certificate validity is requested

CMSBinaryData       goodBefore;                     /* the time in seconds since midnight on 1st
                                                    /* of Jan 1970 before which the
                                                    /* certificate validity is assured
```

```
CMSBinaryData       recheckAfter;       /* the time in seconds since midnight on 1st
                                        /* of Jan 1970 after which the
                                        /* certificate validity should be reconfirmed

/* export declarations */

#if defined(_WINDOWS) && !defined(_WIN32)
#       ifndef WIN16
#               define WIN16
#       endif
#       define EXPT _export
#       define EXPT32
#       define EXPORT _export _far _pascal
#       define EXPORT32
#else
#       if defined(_WIN32) && !defined(_MAC)
#               include <windef.h>
#               define EXPT
#               define EXPT32 _declspec( dllexport )
#               define EXPORT _stdcall
#               define EXPORT32 _declspec( dllexport )
#       else /*not WIN16 or WIN32 */
#               define EXPT
#               define EXPT32
#               define EXPORT
#               define EXPORT32
#       endif
#endif

#endif

/* cmsapi.h - header file for the Certificate Mangement Services Application
Programming Interface  */

#ifndef CMSAPI_H
#define CMSAPI_H
#if !defined(CMSAPI)
#include <cmsbdefs.h>
#else
#include <tkbdefs.h>
#endif
#ifdef _cplusplus
extern “C” {
#endif

EXPORT32 CMSLog EXPORT CMSLog CMSLogin (
```

```
        const char*             cMSVersionNumber,
        const CryptoLibrary     cryptoLibrary,
        CMSBinaryData*          cryptoLibraryPIN,
        const char*             InitializationFile,
        const char*             userID,
        CMSContext*             context );

EXPORT32 CMSLog EXPORT CMSLogout (

        const CMSContext    context );

EXPORT32 void EXPORT CMSEraseKeyHistory (

        const CMSContext    context );

EXPORT 32CMSLog EXPORT CMSGetRecoveredCertificate (

        const CMSContext    context,
        const INT32         index,
        CMSBinaryData*      certificate );

EXPORT 32CMSLog EXPORT CMSGetRecoveredKey (

        const CMSContext    context,
        const INT32         index,
        CMSBinaryData*      privateKey );

EXPORT32 KeyStatus EXPORT CMSKeyStatus (

        const CMSBinaryData*        certificate );

EXPORT32 INT32 EXPORT CMSNumberOfRecoveredKeys (

        const CMSContext    context );

EXPORT 32CMSLog EXPORT CMSRequestKeyRecovery (

        const CMSContext            context,
        const char*                 userReference,
        const CMSBinaryData*        authenticationToken,
        const KeyFlow               keyFlow,
        CMSBinaryData*              certificateAttributes,
        void*                       keyHistory,
        INT32*                      responseTime);

EXPORT 32CMSLog EXPORT CMSRequestKeyUpdate (

        const CMSContext            context,
        const CMSBinaryData*        authenticationToken,
        const CMSBinaryData*        certificateAttributes,
        INT32*                      responseTime);

EXPORT 32CMSLog EXPORT CMSRequestMyCertificate (
```

```
        const CMSContext        context,
        const char*             userReference,
        const CMSBinaryData*    authenticationToken,
        const CMSBinaryData*    registrationForm,
        const CMSBinaryData*    certificateAttributes,
        INT32*                  responseTime);

EXPORT 32CMSLog EXPORT CMSRequestRegistrationForm (

        const CMSContext        context,
        INT32*                  responseTime);

EXPORT 32CMSLog EXPORT CMSRetrieveMyCertificate (

        const CMSContext        context,
        const char*             userReference,
        CMSBinaryData*          authenticationToken,
        const KeyUsage          keyUsage,
        const CMSBinaryData*    policyId,
        KeyPair*                keyPair,
        INT32*                  cATime,
        CMSBinaryData*          cACertificate );

EXPORT 32CMSLog EXPORT CMSRetrieveRecoveredKeys (

        const CMSContext        context,
        const char*             userReference,
        const CMSBinaryData*    authenticationToken,
        const void              keyHistory,
        INT32*                  cATime,
        CMSBinaryData*          cACertificate);

EXPORT 32CMSLog EXPORT CMSRetrieveRegistrationForm (

        const CMSContext        context,
        CMSBinaryData*          registrationForm);

EXPORT 32CMSLog EXPORT CMSRetrieveUpdatedKey (

        const CMSContext        context,
        const CMSBinaryData*    authenticationToken,
        const CMSBinaryData*    oldCertificate
        KeyPair*                keyPair,
        INT32*                  cATime,
        CMSBinaryData*          cACertificate );

EXPORT 32CMSLog EXPORT CMSRevokeMyCertificate (

        const CMSContext        context,
        const CMSBinaryData             certificate,
        INT32*                  responseTime);

EXPORT32 INT32 EXPORT CMSAddName (
```

```
        const CMSContext    context,
        const char*         uniqueName );

EXPORT 32CMSLog EXPORT CMSGetUniqueName (

        const CMSContext    context,
        const INT32         nameIndex,
        char*               uniqueName );

EXPORT 32CMSLog EXPORT CMSRemoveName (

        const CMSContext    context,
        const char*         uniqueName );

EXPORT32 void EXPORT CMSResetNameList (

        const CMSContext    context );

EXPORT 32CMSLog EXPORT CMSGetAttribute (

        const CMSContext    context,
        const INT32         nameIndex,
        const INT32         attributeIndex,
        char*               attribute );

EXPORT 32CMSLog EXPORT CMSGetAttributeValue (

        const CMSContext    context,
        const INT32         nameIndex,
        const INT32         attributeIndex,
        const INT32         valueIndex,
        CMSBinaryData*      attributeValue );

EXPORT32 INT32 EXPORT CMSNumberOfAttributes (

        const CMSContext    context,
        const INT32         nameIndex );

EXPORT32 INT32 EXPORT CMSNumberOfAttributeValues (

        const CMSContext    context,
        const INT32         nameIndex,
        const INT32         attributeIndex);

EXPORT32 INT32 EXPORT CMSNumberOfNames (

        const CMSContext    context );

EXPORT32 void EXPORT CMSReleaseBuffer (

        CMSBinaryData*      buffer);

EXPORT32 CMSLog EXPORT CMSReleaseKeyPair (

        KeyPair*            keyPair);

EXPORT32 CMSLog EXPORT CMSSearchDirectory (
```

```
        const CMSContext    context,
        const char*         searchExpr,
        const char*         searchBase,
        const char*         attribsToReturn ,
        const SearchDepth   searchDepth );

EXPORT32 INT32 EXPORT CMSGetCertificateAttribute (

        const CMSContext    context,
        CMSBinaryData*      attributeType,
        CMSBinaryData*      attributeValue );

EXPORT32 CMSLog EXPORT CMSValidateCertificate (

        const CMSContext            context,
        const CMSBinaryData*        certificateAttributes,
        const CMSBinaryData*        cACertificate,
        const BOOL                  useCMSTime,
        const INT32*                userTime,
        const BOOL                  allowPolicyMapping,
        INT32*                      validUntil );

EXPORT32 void EXPORT CMSGetLogString (

        const CMSLog        log,
        char*               explanation );

EXPORT32 const char* EXPORT CMSLogToString (

        const CMSLog        log );

EXPORT32 BOOL EXPORT CMSQueryLogWarning (

        CMSLog      log );

EXPORT32 const char* EXPORT CMSQueryVersionNumber (void );

#ifdef _cplusplus

}
#endif

#endif

/* cmserr.h - header fiel for the CMS API CMSLog return codes */

#ifndef CMSERR_H

#define CMSERR_H
#define CMS_OK                  0
#define CMS_ERR_START           -3299
#define CMS_ERR_END             -3100
#define CMS_WARN_START          -3099
#define CMS_WARN_END            -3000
```

```
/* Error codes */

enum {
        CMS_AttributesNotPresent                        = -3299,
        CMS_CannotAcceptUserGeneratedPrivateKey         = -3298,
        CMS_CannotConnect                               = -3297,
        CMS_CannotGeneratePublicKey                     = -3296,
        CMS_CannotGetCRL                                = -3295,
        CMS_ContextNotRecognized                        = -3294,
        CMS_CryptoLibraryNotAvailable                   = -3293,
        CMS_CryptoLibraryNotSupported                   = -3292,
        CMS_CryptoLibraryPINIncorrect                   = -3291,
        CMS_DirectoryAccessDenied                       = -3290,
        CMS_EntryNotFound                               = -3289,
        CMS_FileError                                   = -3288,
        CMS_FormattingError                             = -3287,
        CMS_IncorrectAuthenticationToken                = -3286,
        CMS_InvalidAttributeIndex                       = -3285,
        CMS_InvalidAttributeMnemonic                    = -3284,
        CMS_InvalidCertificateSyntax                    = -3283,
        CMS_InvalidFieldName                            = -3282,
        CMS_InvalidListId                               = -3281,
        CMS_InvalidMode                                 = -3280,
        CMS_InvalidNameIndex                            = -3279,
        CMS_InvalidParm                                 = -3278,
        CMS_InvalidPathEntryIndex                       = -3276,
        CMS_InvalidSearchBase                           = -3275,
        CMS_InvalidSearchExpr                           = -3274,
        CMS_InvalidValueIndex                           = -3273,
        CMS_ManagerClientTimeMismatch                   = -3272,
        CMS_MaximumNumberOfOpenContextsExceeded         = -3271,
        CMS_MemoryError                                 = -3270,
        CMS_NoKeyHistoryAvailable                       = -3269,
        CMS_RequiredDataMissing                         = -3268,
        CMS_ResponseNotReady                            = -3267,
        CMS_SecurityProtocolFailure                     = -3266,
        CMS_StateError                                  = -3265,
        CMS_Timeout                                     = -3264,
        CMS_UnexpectedNullPointer                       = -3263,
        CMS_UnknownError                                = -3262,
        CMS_Unsupported                                 = -3261,
        CMS_UserReferenceNotRecognized                  = -3260,
        CMS_UserUniqueNameNotRecognized                 = -3259,
        CMS_VersionNumberNotSupported                   = -3258,
};
```

```
/* Warning codes */
        CMS_SearchSizeLim it                                    = -3099,
        CMS_StringTruncated                                             = -3098,
};
#endif
```