Novell Developer Kit

www.novell.com

October 11, 2006

NOVELL PUBLIC KEY INFRASTRUCTURE TOOLBOX

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Novell Trademarks

For a list of Novell trademarks, see Trademarks (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

The Novell® Public Key Infrastructure Toolbox (NPKIT) API furnishes you with non-directory-centered, public key infrastructure (PKI) services to manage and access X.509 certificates. This API is delivered entirely in the C programming language to provide broad cross-platform support for all platforms that support Novell eDirectory™ (including Solaris*, Linux*, and AIX*), without requiring eDirectory™.

This API will help you use this functionality to further enhance or customize your security solutions without re-writing your own technology. For information about the PKI APIs and how they work together, see NDK: Novell Certificate Server APIs --- Overview.

This guide provides the following topics:

Audience

This guide is intended for Novell Certificate Servers experienced in both the C and Java programming languages..

Feedback

We want to hear your comments and suggestions about this manual. Please use the User Comments feature at the bottom of each page of the online documentation and enter your comments there.

Documentation Updates

For the most recent version of this guide, see Novell Cluster Services SDK (http://developer.novell.com/ndk/ncslib.htm).

Additional Documentation

For documentation on Novell Certificate Server, see the Novell Certificate Server 2.7.x Web site (http://www.novell.com/documentation/crt27/index.html).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

Concepts

1

For more conceptual information about Novell® Public Key Infrastructure Toolbox, see Concepts in Novell Certificate Server™ APIs-Overview.

1.1 NPKIT Dependencies

See Novell eDirectory™ 8.7 System Requirements (http://www.novell.com/products/edirectory/sysreqs.html).

Tasks

2

The Novell® Public Key Infrastructure Toolbox contains functions that enable you to manage and access certificates. Examples of the following implementations can be examined in the NPKIT Sample Code file (../../../samplecode/ncslib_sample/index.htm).

2.1 Getting Started

Before calling many of the Novell PKI Toolbox API functions, you must first create an appropriate context. Call one of these functions:

- NPKIT_x509CreateContext (page 50)
- NPKIT_CRLCreateContext (page 29)

When finished with the operations within one of these contexts, you need to clear the context by calling the corresponding function:

- NPKIT_x509FreeContext (page 59)
- NPKIT_CRLFreeContext (page 38)

2.2 Decoding an X509 Certificate

Basic information about a certificate is obtained by calling NPKIT_x509DecodeCertificate (page 55). Once this function has been called, there are several other functions that can be called to obtain additional certificate information.

1. Create an x509 context by calling NPKIT_x509CreateContext (page 50).
2. Decode the certificate by calling NPKIT_x509DecodeCertificate (page 55).
3. Optionally, to obtain additional information about the certificate:
   - **3a** Obtain key usage information by calling NPKIT_x509KeyUsageInfo (page 65).
   - **3b** Obtain subject alternative names information by calling NPKIT_x509SubjectAltNamesInfo (page 72) to find the number of names, then NPKIT_x509SubjectAltName (page 70)for information about each name.
   - **3c** Obtain issuer alternative names information by calling NPKIT_x509IssuerAltNamesInfo (page 64) to find the number of names, then NPKIT_x509IssuerAltName (page 62) for information about each name.
   - **3d** Obtain information about each extension by calling NPKIT_x509GetExtensionData (page 60) (number of extensions was returned by NPKIT_x509DecodeCertificate (page 55)).
   - **3e** Obtain basic constraint information by calling NPKIT_x509BasicConstraintsInfo (page 49).
   - **3f** Obtain CRL distribution point information by calling NPKIT_x509CRLDistributionPointsInfo (page 54) to find the number of distribution points, then NPKIT_x509CRLDistributionPoint (page 51) for information about each distribution point.

3g Obtain information about any Novell extensions (if any exist) by calling NPKIT_x509NovellExtensionInfo (page 67).

4 Free the x509 context by calling NPKIT_x509FreeContext (page 59).

2.3 Decoding a CRL

Obtain basic information about a CRL by calling NPKIT_CRLDecode (page 30). Once this function has been called, there are several other functions that can be called to obtain additional information about the list. To decode a CRL, follow the steps listed below:

1 Create a CRL context by calling NPKIT_CRLCreateContext (page 29).

2 Decode the CRL by calling NPKIT_CRLDecode (page 30).

3 Optionally, to obtain additional information about the CRL:

3a Obtain information about each list extension by calling NPKIT_CRLExtensionInfo (page 36) (number of list extensions was returned by NPKIT_CRLDecode (page 30)).

3b Obtain basic information about each entry in the list by calling NPKIT_CRLEntryInfo (page 34) (the number of entries in the list was returned by NPKIT_CRLDecode (page 30)). After this function is called, several other functions can be called to obtain additional information about the entry. This is done by using one or more of the following optional methods:

- Obtain information about why the certificate in this entry was revoked by calling NPKIT_CRLReasonCodeInfo (page 40).
- Obtain invalidity date information for the revoked certificate in this entry by calling NPKIT_CRLInvalidityDateInfo (page 39).
- Obtain information about each entry extension by calling NPKIT_CRLEntryExtensionInfo (page 32) (the number of extensions was returned by NPKIT_CRLEntryInfo (page 34)).

4 Free the CRL context by calling NPKIT_CRLFreeContext (page 38).

Functions

3

The Novell® Public Key Infrastructure Toolbox furnishes you with a nondirectory-centered, public key infrastructure to manage and access X.509 certificates. The functions within this document are provided in the C programming language to provide the best cross-platform support for all platforms integrated with Novell eDirectory™. See Dependencies in Chapter 2, “Tasks,” on page 11 for specific configuration and implementation requirements.

Novell Certificate Server™ interfaces, prototypes, and data types are defined in the Novell header files npkit.h, nverify.h, NPKIT_Verify.h, and NPKIT_x509.h. As in all early versions, the API is subject to change.

NOTE: When using this API, all functions should be treated as blocking functions.

NPKIT Error Codes: For a listing of NPKIT error codes, see the Certificate Server “Certificate Server Error Code Constants”.

NPKIT_CacheAddElement

Adds a cache element of type elementType to the context.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheAddElement
(
   NPKIT_CacheContext   context
   const nuint32        elementType,
   const pnuint8        data,
   const nuint32        dataLength,
   void                *reserved1,
   void                *reserved2,
   void                *reserved3,
   void                *reserved4
);
```

Parameters

context

(IN) Defines the context for the function.

elementType

(IN) Specifies one of the following types of cache elements to add:

Value	Description
NPKIT_PEM_OBJ_X509	Specifies an x.509 certificate.
NPKIT_PEM_OBJ_X509_REQUEST	Specifies a PKCS #10 CSR (Certificate Signing Request).
NPKIT_PEM_OBJ_X509_TRUSTED	Specifies a trusted x.509 certificate (that is a CA or root certificate).
NPKIT_PEM_OBJ_CRL	Specifies an x.509 Certificate Revocation List.
NPKIT_PEM_OBJ_PKCS7	Specifies a PKCS #7 file which contains one or more x.509 certificates.
NPKIT_PEM_OBJ_WRAP_PRIV_KEY	Specifies a wrapped private key (that is a private-key that has been cryptographically wrapped for protection).
NPKIT_PEM_OBJ_TERISA_KEY_FILE	Specifies a TERESA key file.
NPKIT_PEM_OBJ_RSA_PRIV_KEY	Specifies a raw RSA private key (PKCS #8).
NPKIT_PEM_OBJ_PRIV_KEY	Specifies a raw private key (PKCS #1).

data

(IN) Specifies the data to be added.

dataLength

(IN) Allocates the length of the data to be added.

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

reserved3

For future use; pass NULL.

reserved4

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheCreateContext (page 19)
NPKIT_CacheFreeContext (page 24)
NPKIT_CacheWrite (page 27)

NPKIT_CacheAddPKCS12Elements

Adds all the elements in the PKCS#12 data to the context.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheAddPKCS12Elements
(
   NPKIT_CacheContext     context
   const unicode         *password
   const pnuint8          pfxData,
   const nuint32          pfxDataLength,
   nuint32               *numberOfElementsAdded,
   void                  *reserved1,
   void                  *reserved2,
   void                  *reserved3,
   void                  *reserved4
);
```

Parameters

context

(IN) Defines the context.

password

(IN) Specifies the password used to decrypt the PKCS12 data.

pfx -

numberOfElementsAdded - .

pfxData

(IN) Specifies the PFX (or PKCS#12) data.

pfxDataLength

(IN) Sets the length of the PFX (or PKCS#12) data.

numberOfElementsAdded

(OUT) Returns the number of elements that were added to the cache.

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

reserved3

For future use; pass NULL.

reserved4

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheAddElement (page 14)
NPKIT_CacheCreateContext (page 19)
NPKIT_CacheFreeContext (page 24)
NPKIT_CacheWrite (page 27)

NPKIT_CacheClearAllElements

.Clears all elements from a cache contex.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheClearAllElements
(
   NPKIT_CacheContext    *context
   void                  *reserved1,
   void                  *reserved2
);
```

Parameters

context

(IN) Specifies the context..

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

See Also

NPKIT_CacheAddElement (page 14)
NPKIT_CacheCreateContext (page 19)
NPKIT_CacheFreeContext (page 24)

NPKIT_CacheCreateContext

Creates a new NPKIT cache context and initializes it with default values.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheCreateContext
(
   NPKIT_CacheContext      *context
);
```

Parameters

context

(IN) The context parameters for the NPKIT cache.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheFreeContext (page 24)

NPKIT_CacheElementInfo

Retreives information about the specified cache element.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheElementInfo
(
   NPKIT_CacheContext   context
   const nuint32        index,
   pnuint32             elementType,
   nuint8 const       **data,
   pnuint32             dataLength,
   pnuint32             flags,
   void                *reserved2,
   void                *reserved3,
   void                *reserved4
);
```

Parameters

context

Points to the newly created cache element context handle.

index

(IN) Specifies which cache element to get information from.

elementType

(OUT) Specifies one of the following types of cache elements to add:

Value	Description
NPKIT_PEM_OBJ_X509	Specifies an x.509 certificate.
NPKIT_PEM_OBJ_X509_REQUEST	Specifies a PKCS #10 CSR (Certificate Signing Request).
NPKIT_PEM_OBJ_X509_TRUSTED	Specifies a trusted x.509 certificate (that is a CA or root certificate).
NPKIT_PEM_OBJ_CRL	Specifies an x.509 Certificate Revocation List.
NPKIT_PEM_OBJ_PKCS7	Specifies a PKCS #7 file which contains one or more x.509 certificates.
NPKIT_PEM_OBJ_WRAP_PRIV_KEY	Specifies a wrapped private key (that is a private-key that has been cryptographically wrapped for protection).
NPKIT_PEM_OBJ_TERISA_KEY_FILE	Specifies a TERESA key file.
NPKIT_PEM_OBJ_RSA_PRIV_KEY	Specifies a raw RSA private key (PKCS #8).

Value	Description
NPKIT_PEM_OBJ_PRIV_KEY	Specifies a raw private key (PKCS #1).
NPKIT_PEM_OBJ_UNKNOWN	Unknown element type.

data

(OUT) Returns the data from the specified cache element.

dataLength

(OUT) Returns the length of the data from the specified cache element.

flags

(IN) Specifies which format the element data will be returned in. Use one of the following:

Value	Description
NPKIT_NORMAL_FORMAT	Specifies that the element info should be returned in the normal or DER format.
NPKIT_PEM_FORMAT	Specifies that the element info should be returned in PEM format.

reserved2

For future use; pass NULL.

reserved3

For future use; pass NULL.

reserved4

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheAddElement (page 14)
NPKIT_CacheCreateContext (page 19)
NPKIT_CacheRead (page 25)

NPKIT_CacheExportToPKCS12

Exports the data in the Cache Context into a Personal Information Exchange Syntax (PFX) format. The cache context must contain a private-key, a matching x.509 certificate, and corresponding certificate chain.
The private key and certificates are encrypted using the input password as specified in the Public Key Cryptography Standards (PKCS) #12.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheExportToPKCS12
(
   NPKIT_CRLContext      context
   const nuint32         flags,
   const unicode        *name,
   const char           *path,
   const unicode        *password,
   const nuint8        **pfxData,
   nuint32              *pfxSize,
   void                 *reserved1,
   void                 *reserved2,
   void                 *reserved3,
   void                 *reserved4
);
```

Parameters

context

(IN) pecifies the NPKIT_Cache context handle for the request. A successful call to NPKIT_CacheCreateContext (page 19) must have previously been made to obtain the context handle.

flags

(IN) Currently no flags are defined; pass a zero value.

name

(IN) Specifies the name of the cache file.

path

(IN) (Optional) If used, specifies the complete file path, including file name to use when reading the cache file. If used, the parameter name should null.

password

(IN) Specifies the password to use to encrypt the data with.

pfxData

(OUT) Returns the PFX data.

pfxSize

(OUT) Returns the size of the PFX data.

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

reserved3

For future use; pass NULL.

reserved4

For future use; pass NULL.

Return Values

Returns 0 if successful or a NICI or PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheCreateContext (page 19)
NPKIT_CacheFreeContext (page 24)

NPKIT_CacheFreeContext

Frees a previously allocated NPKIT cache context and all associated memory.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheFreeContext
(
   NPKIT_CacheContext  context
);
```

Parameters

context

(IN) Specifies context.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheCreateContext (page 19)

NPKIT_CacheRead

Reads the certificate information stored in the cache.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheRead
(
   NPKIT_CacheContext  context
   const nuint32       flags,
   const unicode      *name,
   const char         *path,
   pnuint32            numberOfElements,
   void               *reserved1,
   void               *reserved2
);
```

Parameters

context

(IN) Specifies the NPKIT_Cache context handle for the request. A successful call to NPKIT_CacheCreateContext (page 19) must have previously been made to obtain the context handle.

flags

(IN) Specifies options for reading the cache file. The valid flags and their behavior are listed below:

Value	Description
NPKIT_PEM_NORMAL_NAME	Using the flag NPKIT_PEM_NORMAL_NAME indicates that the parameter name is to be interpreted as a Unicode string, allowing cache files with Unicode names to be stored on file systems which do not support Unicode file names.
NPKIT_PEM_ASCII_NAME	Using the flag NPKIT_PEM_ASCII_NAME indicates that the parameter name is to be interpreted as an ASCII string. (This means that when using this flag, you must pass a null-terminated character string value in the name parameter.)
PKIT_PEM_OVERWRITE_FILE	Enables overwriting of existing cache file.

name

(IN) Specifies the name of the cache file.

path

(IN) (Optional) If used, specifies the complete file path, including file name to use when reading the cache file. If used, the parameter name should NULL.

numberOfElements

(OUT) Returns the number of elements that were added to the cache.

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheCreateContext (page 19)
NPKIT_CacheFreeContext (page 24)
NPKIT_CacheWrite (page 27)

NPKIT_CacheWrite

Writes all the cache elements in the cache context to the specified cache file.

Syntax

```
#include "<npkit.h>, <ntypes.h>"

NWRCODE NPKIT_CacheWrite
(
   NPKIT_CacheContext    context
   const nuint32         flags,
   const unicode        *name,
   const char           *path,
   void                 *reserved1,
   void                 *reserved2
);
```

Parameters

context

(IN) Specifies the NPKIT_Cache context handle for the request. A successful call to NPKIT_CacheCreateContext (page 19) must have previously been made to obtain the context handle.

flags

(IN) flags - Specifies options for writing the cache file. The three possible flags and their behavior are listed below:

Value	Description
NPKIT_PEM_NORMAL_NAME	Using the flag NPKIT_PEM_NORMAL_NAME indicates that the parameter name is to be interpreted as a Unicode string, allowing cache files with Unicode names to be stored on file systems which do not support Unicode file names.
NPKIT_PEM_ASCII_NAME	Using the flag NPKIT_PEM_ASCII_NAME indicates that the parameter name is to be interpreted as an ASCII string. (This means that when using this flag, you must pass a null-terminated character string value in the name parameter.)
NPKIT_PEM_OVERWRITE_FILE	Using the flag NPKIT_PEM_OVERWRITE_FILE specifies that existing cache files with the same name should be overwritten. This flag may be bitwise-OR'ed with either of the two other flags.

name

(IN) Specifies the name of the cache file.

path

(IN) (Optional) If used, specifies the complete file path, including file name to use when writing the cache file. If used, the parameter name should null.

reserved1

For future use; pass NULL.

reserved2

For future use; pass NULL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CacheRead (page 25)

NPKIT_CRLCreateContext

Creates an NPKIT_CRL context handle (formerly NWPKIx509CreateContext).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLCreateContext
(
   NPKIT_CRLContext      *context
);
```

Parameters

context

(OUT) Points to the newly created NPKIT_CRL context handle. This is a nuint32 value.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CRLFreeContext (page 38), NPKIT_CRLDecode (page 30)

NPKIT_CRLDecode

Decodes the specified Certificate Revocation List (CRL) from its ASN.1 DER encoding (formerly NWx509DecodeCRL).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLDecode
(
   NPKIT_CRLContext     context,
   const nuint8        *cRLData,
   nuint32              cRLDataLen,
   nuint8 const       **unsignedCRL,
   pnuint32             unsignedCRLLength,
   pnuint32             signatureAlgorithmType,
   unicode const      **signatureAlgorithmOID,
   nuint8 const       **signature,
   pnuint32             signatureLength,
   pnuint32             version,
   uniocode const     **issuerName,
   struct tm const    **thisUpdate,
   time_t const       **thisUpdateTime,
   struct tm const    **nextUpdate,
   time_t const       **nextUpdateTime,
   pnuint32             numberOfRevokedCertificates,
   pnuint32             numberOfCRLExtensions
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

cRLData

(IN) Points to the CRL to be decoded.

cRLDataLen

(IN) Specifies the size of the data pointed to by cRLData.

unsignedCRL

(OUT) Points to the unsigned portion of the CRL.

unsignedCRLLength

(OUT) Specifies the length of the unsigned portion of the CRL.

signatureAlgorithmType

(OUT) Points to the algorithm used in the signature.

signatureAlgorithmOID

(OUT) Points to a Unicode string that contains a human-readable form of the signature algorithm object identifier (OID) (for example, {1 2 840 113549 1 1 1}).

signature

(OUT) Points to the signature of the CRL.

signatureLength

(OUT) Points to the length of the data in signature.

version

(OUT) Points to the version number of the CRL.

issuerName

(OUT) Points to a Unicode string that contains the name of the Certificate Authority (CA) that issued the CRL.

thisUpdate

(OUT) Points to a `struct tm` representation of the most recent date the CRL was updated. The time is in UTC standard time.

thisUpdateTime

(OUT) Points to a `time_t` representation of the date when the CRL was last updated. Represented as the number of seconds since 00:00:00 UTC January 1, 1970.

nextUpdate

(OUT) Points to a `struct tm` representation of the next date the CRL will be updated. The time is in UTC standard time.

nextUpdateTime

(OUT) Points to a `time_t` representation of the date when the CRL will be updated next. Represented as the number of seconds since 00:00:00 UTC January 1, 1970.

numberOfRevokedCertificates

(OUT) Points to the number of revoked certificates in the CRL.

numberOfCRLExtensions

(OUT) Points to the number of extensions associated with the CRL.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

See Also

NPKIT_CRLCreateContext (page 29), NPKIT_CRLEntryInfo (page 34), NPKIT_CRLExtensionInfo (page 36), NPKIT_CRLFreeContext (page 38)

NPKIT_CRLEntryExtensionInfo

Obtains the specified ASN.1 encoded extension for the CRL entry (formerly NWx509CRLEntryExtensionInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLEntryExtensionInfo
(
   NPKIT_CRLContext           context,
   const nuint32              extensionIndex,
   enum NPKIT_509Extension   *type,
   unicode const            **OID,
   pnbool8                    critical,
   pnuint32                   valueLen,
   nuint8 const             **value
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

extensionIndex

(IN) Specifies which extension is to be returned. `index` is 0 based.

type

(OUT) Points to the extension type of the extension indicated by the `extensionIndex`. See Section 4.10, “X.509 Extensions,” on page 79.

OID

(OUT) Points to a Unicode representation of the OID.

critical

(OUT) Specifies whether the extension is critical or not.

valueLen

(OUT) Specifies the length of data in bytes pointed to by `value`.

value

(OUT) Points to the ASN.1 encoded value of the extension.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must first successfully call NPKIT_CRLDecode (page 30) followed by NPKIT_CRLEntryInfo (page 34). The extension retrieved in this call is for the entry specified in your call to NPKIT_CRLEntryInfo (page 34).

See Also

NPKIT_CRLDecode (page 30), NPKIT_CRLEntryInfo (page 34), NPKIT_CRLInvalidityDateInfo (page 39), NPKIT_CRLReasonCodeInfo (page 40)

NPKIT_CRLEntryInfo

Obtains information about a revoked certificate.

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLEntryInfo
(
   NPKIT_CRLContext              context,
   const nuint32                 index,
   nuint8 const                **serialNumber,
   pnuint32                      serialNumberLen,
   struct tm const             **revocationDate,
   time_t const                **revocationTime,
   pnuint32                      numberOfCRLEntryExtensions
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

index

(IN) Specifies which CRL entry information is to be returned. `index` is 0 based.

serialNumber

(OUT) Points to the serial number of the specified revoked certificate.

serialNumberLen

(OUT) Points to the length of the data in `serialNumber`.

revocationDate

(OUT) Points to a `struct tm` representation of the date the specified certificate was revoked. The time is in UTC standard time.

revocationTime

(OUT) Points to a `time_t` representation of the time the specified certificate was revoked. Represented as the number of seconds since 00:00:00 UTC January 1, 1970.

numberOfCRLEntryExtensions

(OUT) Specifies the number of extensions of the specified certificate.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Returns the serial number, length of the serial number, the date and the time the certificate was revoked for the index specified. A successful call to NPKIT_CRLDecode (page 30) must be made prior to making this call. Subsequent calls to NPKIT_CRLEntryExtensionInfo (page 32) can be made to retrieve the CRL entry extension information for each of the CRL entry extensions identified in numberOfCRLEntryExtensions.

See Also

NPKIT_CRLDecode (page 30), NPKIT_CRLEntryExtensionInfo (page 32), NPKIT_CRLInvalidityDateInfo (page 39), NPKIT_CRLReasonCodeInfo (page 40)

NPKIT_CRLExtensionInfo

Obtains information about the specified extension of the CRL (Certificate Revocation List) (formerly NWx509CRLExtensionInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLExtensionInfo
(
   NPKIT_CRLContext           context,
   nuint32                    index,
   enum NPKIT_x509Extension  *type,
   unicode const            **OID,
   pnbool8                    critical,
   pnuint32                   valueLen,
   nuint8 const             **value
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

index

(IN) Specifies which CRL extension is to be returned. `index` is 0 based.

type

(OUT) Points to the extension type.

OID

(OUT) Points to a Unicode representation of the OID that identifies the extension.

critical

(OUT) Specifies whether the extension is critical or not.

valueLen

(OUT) Specifies the length of the data in `value`.

value

(OUT) Points to the ASN.1 encoded value of the extension indicated by the index.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must call NPKIT_CRLDecode (page 30) successfully.

See Also

NPKIT_CRLDecode (page 30)

NPKIT_CRLFreeContext

Frees a previously allocated CRL context and all associated memory (formerly NWx509FreeContext).

Syntax

```
#include "NPKIT_x509.h"

void NPKIT_CRLFreeContext
(
   NPKIT_CRLContext       context
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle to be freed. This is a nuint32 value.

See Also

NPKIT_CRLCreateContext (page 29)

NPKIT_CRLInvalidityDateInfo

Returns the invalidity date associated with the CRL (Certificate Revocation List) entry (formerly NWx509CRLInvalidityDateInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLInvalidityDateInfo
(
   NPKIT_CRLContext      context,
   struct tm const     **invalidityDate,
   time_t const        **invalidityDateTime
);
```

Parameters

`context`

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

`invalidityDate`

(OUT) Points to a `struct tm` representation of the date the certificate became invalid. The time is in UTC standard time.

`invalidityDateTime`

(OUT) Points to a `time_t` representation of when the certificate became invalid. Represented as the number of seconds since 00:00:00 UTC January 1, 1970.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_CRLDecode (page 30) followed by NPKIT_CRLEntryExtensionInfo (page 32). The invalidity date retrieved in this call is for the entry specified in your call to NPKIT_CRLEntryInfo (page 34). The invalidity date is an optional extension. Therefore, not all CRL entries will have an associated invalidity date.

See Also

NPKIT_CRLDecode (page 30), NPKIT_CRLEntryInfo (page 34), NPKIT_CRLExtensionInfo (page 36)

NPKIT_CRLReasonCodeInfo

Returns the CRL invalidity reason code associated with the CRL entry (formerly NWx509CRLReasonCodeInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_CRLReasonCodeInfo
(
   NPKIT_CRLContext      context,
   pnuint32              reason
);
```

Parameters

context

(IN) Specifies the NPKIT_CRL context handle for the request. This is a nuint32 value.

reason

(OUT) Points to the reason why the certificate is on the CRL. For more information, see Section 4.7, “NPKIT_x509 Certificate Invalidity Reason Flags,” on page 77.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must first successfully call NPKIT_CRLDecode (page 30) followed by NPKIT_CRLEntryInfo (page 34). The reason code retrieved in this call is for the entry specified in your call to NPKIT_CRLEntryInfo (page 34). The reason code is an optional extension. Therefore, not all CRL entries have an associated reason code.

See Also

NPKIT_CRLDecode (page 30), NPKIT_CRLEntryInfo (page 34), NPKIT_CRLExtensionInfo (page 36)

NPKIT_VerifyCertificate

Determines if the specified subjectCertificate can be verified by the issuerCertificate (formerly NWPKIVerifyCertificate).

Syntax

```
#include "NPKIT_Verify.h"

NWRCODE NPKIT_VerifyCertificate
(
   const pnuint8        issuerCertificate,
   const nuint32        issuerCertificateLen,
   const pnuint8        subjectCertificate,
   const nuint32        subjectCertificateLen,
   const pnuint8        CRL,
   const nuint32        CRLLen,
   pnuint32             reason,
   pnuint32             holdInstruction,
   void                *reserved1,
   void                *reserved2,
   void                *reserved3,
   void                *reserved4
);
```

Parameters

issuerCertificate

(IN) Points to the DER encoded X.509 certificate to use to verify the subject certificate.

issuerCertificateLen

(IN) Specifies the size of the issuer certificate.

subjectCertificate

(IN) Points to the DER encoded X.509 subject certificate to verify.

subjectCertificateLen

(IN) Specifies the size of the subject certificate.

CRL

(IN) Specifies the DER encoded CRL. (Not implemented in this release; pass in NULL.)

CRLLen

(IN) Specifies the size of the CRL. (Not implemented in this release; pass in NULL.)

reason

(OUT) If the certificate is invalid, otherwise specifies the reason code. See Section 4.7, “NPKIT_x509 Certificate Invalidity Reason Flags,” on page 77.

holdInstruction

(OUT) If the certificate has been revoked, and the reason code is *certificateHold*, otherwise specifies the hold instruction from the CRL. (Not implemented in this release; pass in NULL.)

reserved1

Reserved for future use. Pass in NULL.

reserved2

Reserved for future use. Pass in NULL.

reserved3

Reserved for future use. Pass in NULL.

reserved4

Reserved for future use. Pass in NULL.

Return Values

Returns 0 if successful or a PKI or NICI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

In this release NPKIT_VerifyCertificate (page 28) checks the following:

- Issuer and subject names agree.
- Subject validity dates are a subset of issuer validity dates.
- Validity dates are valid.
- The issuer certificate signed the subject certificate.
- The issuer is a CA.
- The path length constraints have not been exceeded.
- The key usage of issuer allows for certificate signing.
- The issuer's critical extensions are supported

This function does not check for certificate revocation. Use NPKIT_VerifyCertChain (page 43) for complete certificate verification

See Also

NPKIT_VerifyCertChain (page 43), NPKIT_VerifyCertChainWithCallback (page 46)

NPKIT_VerifyCertChain

Verifies the certificate chain passed in (formerly NWPKIVerifyCertChain).

Syntax

```
#include "NPKIT_Verify.h"

NWRCODE NPKIT_VerifyCertChain
(
   NPKI_CertChain            *certificates,
   nuint32                    flags,
   pnuint32                   cRLReason,
   pnuint32                   cRLHoldInstruction,
   time_t                    *cRLRevocationTime,
   time_t                    *cRLInvalidityDateTime,
   NPKI_CertChain           **revokedCertificate,
   pnuint32                   certInvalidityReason,
   NPKIT_CRLCacheContext      CRLCacheContext,
   void                      *reserved1,
   void                      *reserved2,
   void                      *reserved3,
   void                      *reserved4
);
```

Parameters

`certificates`

(IN) Points to the linked list of NPKI_CertChain structures, each of which contains an X.509 DER encoded certificate which is to be verified. The linked list of certificates must be in leaf to root order and the last certificate, in the list, is assumed to be a trusted certificate. (If the last certificate has a CRL Distribution Point extension, it must be a self-signed certificate.)

`flags`

(IN) Specifies whether to verify the certificate, certificate revocation, both, or neither. Use one or more of the following flags:

NPKI_VERIFY_NORMAL
NPKI_VERIFY_DONT_CHECK_CERTIFICATE
NPKI_VERIFY_DONT_CHECK_CRL

For more information, see Section 4.5, “NPKI_VerifyCallBackStruct Flag Values,” on page 75.

`cRLReason`

(OUT) If the certificate has been revoked, specifies the reason code from the CRL (that is, private key compromised, affiliate change, superseded, etc.). This parameter is only set if the return code is PKI_E_CERT_INVALID and the `certInvalidityReason` is set to NPKIx509Invalid_Certificate_On_CRL. For more information, see Section 4.7, “NPKIT_x509 Certificate Invalidity Reason Flags,” on page 77.

`cRLHoldInstruction`

(OUT) If the certificate has been revoked, specifies the hold instruction from the CRL. This parameter is only set if the return code is PKI_E_CERT_INVALID, the *certInvalidityReason* is set to NPKIx509Invalid_Certificate_On_CRL, and the *cRLReason* is set to PKI_CERTIFICATE_HOLD. The possible values for *cRLHoldInstruction* are:

PKI_HOLD_INSTRUCTION_NONE
PKI_HOLD_INSTRUCTION_CALL_ISSUER
PKI_HOLD_INSTRUCTION_REJECT

For more information, see Section 4.9, "NPKIT_x509 CRL Hold Types," on page 79.

cRLRevocationTime

(OUT) Points to the time the certificate became invalid. This parameter only set if the return code is PKI_E_CERT_INVALID and the *certInvalidityReason* is set to NPKIx509Invalid_Certificate_On_CRL.

cRLInvalidityDateTime

(OUT) Points to the time the CRL becomes invalid.

revokedCertificate

(OUT) Points to the node in the linked list of NPKI_CertChain structures that contains the invalid certificate. This parameter only set if the return code is PKI_E_CERT_INVALID.

certInvalidityReason

(OUT) Reason why the certificate is invalid. This will only be set if the return code is set to PKI_E_CERT_INVALID.

CRLCacheContext

(OUT) Reserved for future use. Pass in NULL.

reserved1

(OUT) Reserved for future use. Pass in NULL.

reserved2

(OUT) Reserved for future use. Pass in NULL.

reserved3

(OUT) Reserved for future use. Pass in NULL.

reserved4

(OUT) Reserved for future use. Pass in NULL.

Return Values

Returns 0 if successful or a NICI or PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

The certificate chain must be in leaf to root order. The last certificate in the chain is assumed to be trusted. If any certificate is invalid (that is, revoked or expired), this function returns an error. You must allocate and deallocate all the certificate chain structures and data.

See Also

NPKIT_VerifyCertificate (page 41), NPKIT_VerifyCertChainWithCallback (page 46)

NPKIT_VerifyCertChainWithCallback

Creates a thread that verifies the certificate chain passed in (formerly NWPKIVerifyCertChainWithCallback).

Syntax

```
#include "NPKIT_Verify.h"

NWRCODE NPKIT_VerifyCertChainWithCallback
(
   NPKI_VerifyCallBackStruct    *data
);
```

Parameters

data

(IN) Points to a NPKI_VerifyCallBackStruct structure. For more information, see Section 4.5, "NPKI_VerifyCallBackStruct Flag Values," on page 75. You must set the first three fields in this structure and the rest of the fields can be filled in by calling that NPKIT_VerifyCertChainWithCallback (page 46) function. You must allocate and deallocate this structure.

Return Values

This function returns 0 if successful, or a NICI error or a platform-specific error code if a thread could not be created. The return value from the verification is returned in the code field of the NPKI_VerifyCallBackStruct structure.

Remarks

The certificate chain must be in leaf to root order. The last certificate in the chain is assumed to be trusted. If any certificate is invalid (that is, revoked, or expired), the code field in the Section 4.5, "NPKI_VerifyCallBackStruct Flag Values," on page 75 contains an error. You must allocate and deallocate the certificate chain structures and data and provide a callback function to receive the results of the verification.

See Also

NPKIT_VerifyCertificate (page 41), NPKIT_VerifyCertChain (page 43), NPKIT_VerifyIssuerSubjectNameMatch (page 47)

NPKIT_VerifyIssuerSubjectNameMatch

Verifies whether the `subjectCerficate's` issuer name matches the `issuerCertificate's` subject name.

Syntax

```
#include "NPKIT_Verify.h"

NWRCODE NPKIT_VerifyIssuerSubjectNameMatch
(
   const pnuint8       issuerCertificate,
   const nuint32       issuerCertificateLen,
   const pnuint8       subjectCertificate,
   const nuint32       subjectCertificateLen
);
```

Parameters

`issuerCertificate`

(IN) Points to the DER encoded issuer certificate to use to verify the subject certificate.

`issuerCertificateLen`

(OUT) Specifies the length of *issuerCertificate*.

`subjectCertificate`

(IN) Points to the DER encoded subject certificate to verify.

`subjectCertificateLen`

(IN) Specifies the length of the subject certificate.

Return Values

Returns 0 if names match, PKI_E_SUBJECT_NAME_COMPARISON_FAILURE if the names do not match, or a NICI or other PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

This is a lightweight function that can be used to construct and sort a certificate chain.

NPKIT_Version

Checks the version number of the NPKIT library being used.

Syntax

```
#include "NPKIT_x509.h"

 or #include "NPKIT_Verify.h"

nuint32 NPKIT_Version(void);
```

Return Values

Returns the version of NPKIT as an array of four bytes, the first two bytes being the major version number, and the second two bytes being the minor version number. See Section 4.6, “NPKIT_Version Values,” on page 76.

NPKIT_x509BasicConstraintsInfo

Retrieves the details about the Basic Constraints extension, if the extension exists on the certificate (formerly NWx509BasicConstraintsInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509BasicConstraintsInfo
(
   NPKIT_x509Context     context,
   nuint8 const         **cA,
   nuint32 const        **pathLenConstraint
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

cA

(OUT) Points to the value *cA* as encoded in the extension. If this value is zero, the certificate is not for a CA. If the value is non-zero, the certificate is for a CA. For more information, see the Section 4.1, “Basic Constraints Extension Values,” on page 73.

pathLenConstraint

(OUT) Points to the value *pathLenConstraint* as encoded in the extension. This value represents the number of levels of CAs that this Certificate Authority is authorized to create. The value -1 is returned if there is no constraint. For more information, see the Section 4.1, “Basic Constraints Extension Values,” on page 73.

Return Values

Returns 0 if successful, or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must first successfully call NPKIT_x509BasicConstraintsInfo (page 49). The Basic Constraints extension is an optional extension. Therefore, not all certificates have Basic Constraints information.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509GetExtensionData (page 60)

NPKIT_x509CreateContext

Creates a new NPKIT_x509 API context handle (formerly NWx509CreateContext).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509CreateContext
(
   NPKIT_x509Context      *context
);
```

Parameters

context

(OUT) Points to the newly created NPKIT_x509 context handle. This is a nuint32 value.

Return Values

This routine returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

This context is used for ASN.1 decoding of X.509 objects.

See Also

NPKIT_x509FreeContext (page 59)

NPKIT_x509CRLDistributionPoint

Returns information about a CRL Distribution Point in the certificate (formerly NWx509CRLDistributionPoint).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509CRLDistributionPoint
(
   NPKIT_x509Context     context,
   nuint32               index,
   pnuint32              dataSets,
   nuint8               *fullNameType,
   nuint8 const        **fullNameValue,
   pnuint32              fullNameValueLength,
   unicode const       **fullName,
   unicode const       **nameRelativeToCRLIssuer,
   pnuint32              reasons,
   nuint8               *crlIssuerType,
   nuint8 const        **crlIssuerValue,
   pnuint32              crlIssuerValueLength,
   unicode const       **crlIssuerName
);
```

Parameters

`context`

(IN) Specifies the NPKIT_x509 context handle for the request.This is a nuint32 value.

`index`

(IN) Specifies which CRL distribution point is to be returned. `index` is 0 based.

`dataSets`

(OUT) Specifies which sets of data the function has returned. For more information, see Section 4.2, “CRL Distribution Point Values,” on page 74.

`fullNameType`

(OUT) Points to the type of the ASN1 NAME in the certificate distribution point (CDP) (for example, URI, Directory Name, etc.). This parameter only set if *dataSets* includes the value Section 4.2, “CRL Distribution Point Values,” on page 74.

`fullNameValue`

(OUT) Points to where the CRL can be acquired as encoded in the certificate. This parameter only set if *dataSets* includes the value NPKIT_x509DistPtsFullName.

`fullNameValueLength`

(OUT) Specifies the length in bytes of the data that *fullNameValue* points to. This parameter will only be set if *dataSets* includes the value NPKIT_x509DistPtsFullName.

fullName

(OUT) Points to a Unicode string containing the human-readable representation of where the CRL can be acquired. This parameter only is only set if the *fullName* type can be converted to a human-readable representation and if *dataSets* includes the value NPKIT_x509DistPtsFullName.

nameRelativeToCRLIssuer

(OUT) Points to where the CRL can be acquired, relative to the CRL issuer name as encoded in the certificate. Set this parameter only if *dataSets* includes NPKIT_x509DistPtsNameRelativeToCRLIssuer.

reasons

(OUT) Specifies the ivalidity reasons contained in the CRL. See Section 4.7.1, "NPKIT_x509 CRL Distribution Point Reason Code," on page 78.

crlIssuerType

(OUT) Points to the Issuer Name type; for example, X.500, distinguished name (DN), rfc822, DNS name, IP address, or URL. This parameter is only set if *dataSets* includes the value NPKIT_x509DistPtsCRLIssuer.

crlIssuerValue

(OUT) Points to the Issuer Name as encoded in the certificate. This parameter is only set if *dataSets* includes the value NPKIT_x509DistPtsCRLIssuer.

crlIssuerValueLength

(OUT) Specifies the length of the data pointed to by `crlIssuerValue`. This parameter is only set if `dataSets` includes the value NPKIT_x509DistPtsCRLIssuer.

crlIssuerName

(OUT) Points to a Unicode string containing the human-readable representation of the CRL Issuer's Name. This parameter only is set if the *crlIssuerName* type can be converted to a human-readable representation and if *dataSets* includes the value NPKIT_x509DistPtsCRLIssuer.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55) before calling NPKIT_x509CRLDistributionPointsInfo (page 54). Calling NPKIT_x509CRLDistributionPointsInfo is not necessary, but should be done before calling this function to determine whether distribution points exist, and how many there are. You can pass in a NULL to any of the OUT parameters, in which case no value is returned for that parameter. The CRL distribution points extension is an optional extension. Therefore, not all certificates have CRL distribution points information.

See Also

NPKIT_x509CRLDistributionPointsInfo (page 54), NPKIT_x509DecodeCertificate (page 55)

NPKIT_x509CRLDistributionPointsInfo

Obtains the number of CRL distribution points encoded in the certificate (formerly NWx509CRLDistributionPointsInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509CRLDistributionPointsInfo
(
   NPKIT_x509Context    context,
   pnuint32             numCRLDistributionPoints
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

numCRLDistributionPoints

(OUT) Specifies the number of CRL distribution points encoded in the certificate.

Return Values

Returns 0 if successful, or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). The CRL distribution points extension is an optional extension. Therefore, not all certificates have CRL distribution points information.

See Also

NPKIT_x509CRLDistributionPoint (page 51), NPKIT_x509DecodeCertificate (page 55)

NPKIT_x509DecodeCertificate

Decodes the specified certificate from its ASN.1 DER encoding (formerly NWx509DecodeCertificate).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509DecodeCertificate
(
   NPKIT_x509Context     context,
   const nuint8         *certificate,
   const nuint32         certificateLen,
   nuint8 const        **unsignedCertificate,
   pnuint32              unsignedCertificateLen,
   nuint8 const        **signature,
   pnuint32              signatureLen,
   nuint8 const        **serialNumber,
   pnuint32              serialNumberLen,
   nuint8 const        **keyModulus,
   pnuint32              keyModulusLen,
   nuint8 const        **keyExponent,
   pnuint32              keyExponentLen,
   unicode const       **publicKeyAlgorithmOID,
   unicode const       **signatureKeyAlgorithmOID,
   unicode const       **subjectName,
   unicode const       **issuerName,
   struct tm const     **startDate,
   struct tm const     **endDate,
   time_t const        **startTime,
   time_t const        **endTime,
   pnuint32              numberOfExtensions
   pnuint32              version
   pnuint32              keySize
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

certificate

(IN) Points to the DER encoded X.509 certificate you want to be ASN.1 decoded.

certificateLen

(IN) Specifies the size of the certificate.

unsignedCertificate

(OUT) Points to the unsigned certificate. This is a pointer to the start of the *tbsCertificate* field of the ASN.1 object *Certificate*. (The unsigned portion of the certificate should be used along with the signature to validate that the data in the certificate has not been modified or corrupted.)

unsignedCertificateLen

(OUT) Specifies the ASN.1 encoded value for the size of `unsignedCertificate` This is the size of the data returned in unsignedCertificate (that is, the size of the `tbsCertificate` field of the ASN.1 object Certificate).

signature

(OUT) Points to the start of the *signatureValue* field of the ASN.1 object *Certificate*. The unsigned portion of the certificate should be used along with the signature to validate that the data in the certificate has not been modified or corrupted.

signatureLen

(OUT) Specifies the ASN.1 encoded value for the size of `signature`. This is the size of the data returned in signature (that is, the size of the `signatureValue` field in the ASN.1 object `Certificate`).

serialNumber

(OUT) Points to the serial number.

serialNumberLen

(OUT) Specifies the length of *serialNumber*.

keyModulus

(OUT) Points to the key modulus.

keyModulusLen

(OUT) Specifies the length of *keyModulus*.

keyExponent

(OUT) Points to the key exponent.

keyExponentLen

(OUT) Specifies the length of *keyExponent*.

publicKeyAlgorithmOID

(OUT) Points to a Unicode string that contains a human-readable representation of the public key algorithm OID (for example, {1 2 840 113549 1 1 1}).

signatureKeyAlgorithmOID

(OUT) Points to a Unicode string that contains a human-readable representation of the signature key algorithm OID (for example, {1 2 840 113549 1 1 5}).

subjectName

(OUT) Points to a Unicode string representing the subject name.

issuerName

(OUT) Points to a Unicode string representing the issuer name.

startDate

(OUT) Points to a *struct tm* representation of the *validFrom* portion of the *Validity*. The time is in UTC standard time.

endDate

(OUT) Points to a *struct tm* representation of the *validTo* portion of the *Validity*. The time is in UTC standard time.

startTime

(OUT) Points to a *time_t* representation of the *validFrom* portion of the *Validity*. This time is represented as the number of seconds since 00:00:00 UTC January 1, 1970.

endTime

(OUT) Points to a *time_t* representation of the *validTo* portion of the *Validity*. This time is represented as the number of seconds since 00:00:00 UTC January 1, 1970.

numberOfExtensions

(OUT) Specifies the number of extensions encoded in the certificate.

version

(OUT) Specifies the version of the certificate.

keySize

(OUT) Specifies the key size of the public key in the certificate.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

After calling this function successfully, iterative calls to NPKIT_x509GetExtensionData (page 60) can be made to retrieve the ASN.1 encoded certificate extensions. Also, if the corresponding extensions exist in the certificate, you can call the following functions to get the decoded specifics about the extension:

NPKIT_x509BasicConstraintsInfo (page 49)
NPKIT_x509SubjectAltNamesInfo (page 72)
NPKIT_x509IssuerAltNamesInfo (page 64)
NPKIT_x509KeyUsageInfo (page 65)
NPKIT_x509CRLDistributionPointsInfo (page 54)
NPKIT_x509NovellExtensionInfo (page 67)

See Also

NPKIT_x509BasicConstraintsInfo (page 49), NPKIT_x509CRLDistributionPointsInfo (page 54), NPKIT_x509GetExtensionData (page 60), NPKIT_x509IssuerAltNamesInfo (page 64),

NPKIT_x509KeyUsageInfo (page 65), NPKIT_x509NovellExtensionInfo (page 67), NPKIT_x509SubjectAltNamesInfo (page 72)

NPKIT_x509FreeContext

Frees a previously allocated NPKIT_x509 context and all associated memory (formerly NWx509FreeContext).

Syntax

```
#include "NPKIT_x509.h"

void NPKIT_x509FreeContext
(
   NPKIT_x509Context      context
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle to be freed. This is a nuint32 value.

See Also

NPKIT_x509CreateContext (page 50)

NPKIT_x509GetExtensionData

Retrieves the ASN.1 encoded certificate extension specified by *index* (formerly NWx509GetExtensionData).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509GetExtensionData
(
   NPKIT_x509Context             context,
   nuint32                       index,
   enum NPKIT_x509Extension      type,
   unicode const               **OID,
   pnbool8                       critical,
   pnuint32                      valueLen,
   nuint8 const                **value
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context for the request. This is a nuint32 value.

index

(IN) Specifies which extension is to be returned. `index` is 0 based.

type

(OUT) Specifies the type of extension. For more information, see the *NPKIT_x509Extension* section.

OID

(OUT) Points to a Unicode string that contains a human-readable representation of the OID.

critical

(OUT) Specifies whether the extension is critical or not.

valueLen

(OUT) Specifies the length of the value.

value

(OUT) Points to the ASN.1 encoded extension.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). If the extension is of the corresponding type, you can call one of the following functions to retrieve the decoded extension information:

NPKIT_x509BasicConstraintsInfo (page 49)
NPKIT_x509SubjectAltNamesInfo (page 72)
NPKIT_x509IssuerAltNamesInfo (page 64)
NPKIT_x509KeyUsageInfo (page 65)
NPKIT_x509CRLDistributionPointsInfo (page 54)
NPKIT_x509NovellExtensionInfo (page 67)

See Also

NPKIT_x509BasicConstraintsInfo (page 49), NPKIT_x509CRLDistributionPointsInfo (page 54), NPKIT_x509DecodeCertificate (page 55), NPKIT_x509IssuerAltNamesInfo (page 64), NPKIT_x509KeyUsageInfo (page 65), NPKIT_x509NovellExtensionInfo (page 67), NPKIT_x509SubjectAltNamesInfo (page 72)

NPKIT_x509IssuerAltName

Retrieves the specified Issuer Alternative Name and related information (formerly NWx509IssuerAltName).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509IssuerAltName
(
   NPKIT_x509Context     context,
   nuint32               index,
   nuint8               *type,
   nuint8 const        **value,
   pnuint32              length,
   unicode const        **name
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

index

(IN) Specifies which issuer alternative name is to be returned. `index` is 0 based.

type

(OUT) Points to the type of issuer alternative name. For more information, see Section 4.3, "General Name Type Extensions," on page 74.

value

(OUT) Points to the ASN.1 encoded issuer alternative name.

length

(OUT) Specifies the length of *value*.

name

(OUT) Points to a Unicode string representation of the issuer alternative name.

NOTE: This field is only set when the issuer alternative name is one of the following extension types:

X509_GENERAL_NAME_RFC822_NAME
X509_GENERAL_NAME_DNS_NAME
X509_GENERAL_NAME_DIRECTORY_NAME

For more details, see Section 4.3, "General Name Type Extensions," on page 74.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). You should call NPKIT_x509IssuerAltNamesInfo (page 64) before calling this function to determine how many issuer alternative names are encoded in the certificate. For more details about issuer alternative names, see Section 4.3, "General Name Type Extensions," on page 74. The issuer alternative names extension is optional. Therefore, not all certificates have issuer alternative names.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509IssuerAltNamesInfo (page 64), NPKIT_x509SubjectAltName (page 70)

NPKIT_x509IssuerAltNamesInfo

Retrieves the number of issuer alternative names encoded in the certificate if the issuer alternative names extension exists in the certificate (formerly NWx509IssuerAltNamesInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509IssuerAltNamesInfo
(
   NPKIT_x509Context      context,
   pnuint32               numIssuerAltNames
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

numIssuerAltNames

(OUT) Specifies the number of issuer alternative names encoded in the certificate.

Return Values

This routine returns 0 if successful, or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). Iterative calls to NPKIT_x509IssuerAltName (page 62) can be made to retrieve each of the issuer alternative names. The issuer alternative names extension is optional. Therefore, not all certificates will have Issue Alternative Names.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509GetExtensionData (page 60), NPKIT_x509IssuerAltName (page 62)

NPKIT_x509KeyUsageInfo

Retrieves the information from the key usage extension of the certificate if a key usage extension is encoded in the certificate (formerly NWx509KeyUsageInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509KeyUsageInfo
(
   NPKIT_x509Context     context,
   pnuint16               keyUsage
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

keyUsage

(OUT) Specifies the key usage value as encoded in the certificate. The following defines can be used with a bit-wise AND to determine which key usages are encoded.

Constant	Value
X509_KEY_USAGE_DIGITAL_SIGNATURE	0x8000
X509_KEY_USAGE_NON_REPUDIATION	0x4000
X509_KEY_USAGE_KEY_ENCIPHERMENT	0x2000
X509_KEY_USAGE_DATA_ENCIPHERMENT	0x1000
X509_KEY_USAGE_KEY_AGREEMENT	0x0800
X509_KEY_USAGE_KEY_CERT_SIGN	0x0400
X509_KEY_USAGE_CRL_SIGN	0x0200
X509_KEY_USAGE_ENCIPHER_ONLY	0x0100
X509_KEY_USAGE_DECIPHER_ONLY	0x0080

For more information about key usages, see Section 4.4, “Key Usage Extension Values,” on page 75.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see “Certificate Server Error Code Constants”.

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). The key usage extension is optional. Therefore, not all certificates will have key usages.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509GetExtensionData (page 60)

NPKIT_x509NovellExtensionInfo

Retrieves the Novell Security Attribute extension information encoded in the certificate if the Novell Security Attribute extension exists in the certificate.

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509NovellExtensionInfo
(
   NPKIT_x509Context    context,
   unicode const      **version,
   unicode const      **URIReference,
   nbool8              *keyQEnforceQuality,
   nint16              *keyQCSCriteria,
   nint16              *keyQCSRating,
   nint16              *keyQCryptoCriteria,
   nint16              *keyQCryptoRating,
   nint16              *keyQKeyStorage,
   nbool8              *cryptoProEnforceQuality,
   nint16              *cryptoProCSCriteria,
   nint16              *cryptoProCSRating,
   nint16              *cryptoProCryptoCriteria,
   nint16              *cryptoProCryptoRating,
   nint16              *cryptoProKeyStorage,
   nint16              *certificateClass,
   nuint8 const       **EIDRootLabel,
   nint32              *EIDRootLabelLen,
   nuint8 const       **EIDEnterpriseLabel,
   nuint8 const        *EIDEnterpriseLabelLen,
   nuint8 const       **EIDRegistryLabel,
   nint32              *EIDRegistryLabelLen
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

version

(OUT) Points to a Unicode string containing the version of the Novell Security Attribute.

URIReference

(OUT) Points to a Unicode string containing a URI where more information about the Novell Security Attributes can be found.

keyQEnforceQuality

(OUT) Points to the enforce quality flag, which specifies whether the cryptography provider can use the private key on a platform that does not meet the minimum key quality attributes specified.

keyQCSCriteria

(OUT) Points to the computer security criteria under which the machine used to generate the key pair was evaluated (for example, TCSEC or Common Criteria).

keyQCSRating

(OUT) Points to the computer security rating of the machine used to generate the key pair (for example, TCSEC C2 EVALUATED).

keyQCryptoCriteria

(OUT) Points to the cryptographic module criteria under which the machine used to generate the key pair was evaluated (for example, FIPS 140-1).

keyQCryptoRating

(OUT) Points to the cryptographic module rating of the machine used to generate the key pair (for example, FIPS 140-1 VENDOR INSPECTED).

keyQKeyStorage

(OUT) Points to the key storage quality which represents the protection used to secure the private key (for example, password, biometric).

cryptoProEnforceQuality

(OUT) Points to the enforce quality flag, which specifies whether the user will use the private key on a platform that meets the minimum cryptography process attributes specified.

cryptoProCSCriteria

(OUT) Points to the computer security criteria under which the machine that uses the private key was evaluated (that is, TCSEC or Common Criteria).

cryptoProCSRating

(OUT) Points to the cryptographic module rating of the machine that uses the private key (that is, FIPS 140-1 VENDOR INSPECTED).

cryptoProCryptoCriteria

(OUT) Points to the cryptographic module criteria under which the machine that uses the private key was evaluated (that is, FIPS 140-1).

cryptoProCryptoRating

(OUT) Points to the cryptographic module rating of the machine that uses the private key (that is, FIPS 140-1 VENDOR INSPECTED).

cryptoProKeyStorage

(OUT) Points to the cryptography process storage quality which represents the protection used to secure the private key (for example, password, biometric).

certificateClass

(OUT) Points to the certificate class, which represents the amount of due diligence preformed by the CA before signing the certificate (for example, e-mail address, enterprise name, government agency).

EIDRootLabel

(OUT) Points to the enterprise identifier, which specifies the levels and categories for secrecy and integrity for the Root authority.

EIDRootLabelLen

(OUT) Specifies the length of the `EIDRootLabel` field.

EIDEnterpriseLabel

(OUT) Points to the enterprise identifier, which specifies the levels and categories for secrecy and integrity for the enterprise authority.

EIDEnterpriseLabelLen

(OUT) Points to the length of the *EIDEnterpriseLabel* field.

EIDRegistryLabel

(OUT) Points to the enterprise identifier which specifies the levels and categories for secrecy and integrity for the registry authority.

EIDRegistryLabelLen

(OUT) Points to the length of the *EIDRegistryLabel* field.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). The Novell Security Attribute contains information about the cryptographic key quality and operating system's security assurance.For more information about Novell Security Attributes, see Novell Certificate Extension Attributes - Novell Security Attributes (http://developer.novell.com/repository/attributes/pkisv10.pdf). The Novell Security Attributes extension is optional. Therefore, not all certificates have Novell Security Attributes

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509GetExtensionData (page 60)

NPKIT_x509SubjectAltName

Retrieves the specified subject alternative name and related information (formerly NW509SubjectAltName).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509SubjectAltName
(
   NPKIT_x509Context      context,
   nuint32                index,
   nuint8                 *type,
   nuint8 const           **value,
   pnuint32               length,
   unicode const          **name
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

index

(IN) Specifies which subject alternative name is to be returned. `index` is 0 based.

type

(OUT) Points to the type of subject alternative name. For more information, see the section Section 4.3, “General Name Type Extensions,” on page 74.

value

(OUT) Points to the ASN.1 encoded subject alternative name.

length

(OUT) Specifies the length of *value*.

name

(OUT) Points to a Unicode string containing a representation of the subject alternative name.

NOTE: This field is only set when the Subject Alternative Name is one of the following types:

X509_GENERAL_NAME_RFC822_NAME
X509_GENERAL_NAME_DNS_NAME
X509_GENERAL_NAME_DIRECTORY_NAME

For more details, see Section 4.3, “General Name Type Extensions,” on page 74.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). You should call NPKIT_x509SubjectAltNamesInfo (page 72) before calling this function to determine how many subject alternative names are encoded in the certificate. For more details about subject alternative names, see Section 4.3, "General Name Type Extensions," on page 74. The subject alternative name extension is optional. Therefore, not all certificates have subject alternative names.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509GetExtensionData (page 60), NPKIT_x509IssuerAltName (page 62), NPKIT_x509SubjectAltNamesInfo (page 72)

NPKIT_x509SubjectAltNamesInfo

Retrieves the number of subject alternative names encoded in the certificate if the subject alternative names extension exists in the certificate (formerly NWx509SubjectAltNamesInfo).

Syntax

```
#include "NPKIT_x509.h"

NWRCODE NPKIT_x509SubjectAltNamesInfo
(
   NPKIT_x509Context     context,
   pnuint32              numSubjectAltNames
);
```

Parameters

context

(IN) Specifies the NPKIT_x509 context handle for the request. This is a nuint32 value.

numSubjectAltNames

(OUT) Specifies the number of subject alternative names encoded in the certificate.

Return Values

Returns 0 if successful or a PKI error code if not successful. For a listing of NPKIT error codes, see "Certificate Server Error Code Constants".

Remarks

Before calling this function, you must first successfully call NPKIT_x509DecodeCertificate (page 55). Iterative calls to NPKIT_x509SubjectAltName (page 70) can be made to retrieve each of the subject alternative names.

The subject alternative name extension is optional. Therefore, not all certificates have subject alternative names.

See Also

NPKIT_x509DecodeCertificate (page 55), NPKIT_x509SubjectAltName (page 70)

Values

4

Novell Certificate Server interfaces, prototypes, and data types are defined in the Novell `npkit.h`, `NPKIT_Verify.h`, `NPKIT_x509.h`, `npki_ver.h`, and `nverify.h` header files. This section contains the following topics:

4.1 Basic Constraints Extension Values

The X.509 Basic Constraints extension is used to specify that a certificate belongs to a CA. The X.509 Basic Constraints extension has essentially two parts:

- *CA*: Specifies whether the certificate is for a CA.
- *pathLenConstraint*: If the certificate is for a CA, *pathLenConstraint* specifies how many subordinate levels of a certificate chain that the CA can certify. The *pathLenConstraint* can range from zero, meaning that the CA cannot certify other CAs but can certify leaf objects (that is, user and server certificates) to infinite (that is, when *pathLenConstraint* is not specified).

Certificates for CAs must have the Basic Constraints extension encoded. Other certificates should not.

The Basic Constraints extension uses the general-purpose extension structure *NPKI_Extension* described in Section 4.10, “X.509 Extensions,” on page 79.

There is one value specific flag defined for the Basic Constraints extension:

Value	Name	Description
0xffffffff	X509_CA_PATH_LENGTH_UNLIMITED	Compare this value with the value returned in *pathLenConstraint* from NPKIT_x509BasicConstraintsInfo (page 49) to determine if the CA path length is unlimited

4.2 CRL Distribution Point Values

Any number of the following defined values will be returned in the `datasets` field by a successful call to NPKIT_x509CRLDistributionPoint (page 51).

Value	Name	Description
0x0001	NPKIT_x509DistPtsFullName	The full name of the distribution point is being passed back.
0x0002	NPKIT_x509DistPtsNameRelativeToCRLIssuer	The distribution points name is relative to the CRL issuer.
0x0004	NPKIT_x 509DistPtsReasons	The reason codes are being passed back.
0x0008	NPKIT_x 509DistPtsCRLIssue	The CRL issuer name is being passed back.

4.3 General Name Type Extensions

The following general name type values specify which encoding format is used to encode the general name:

Value	Name	Description
0x0000	X509_GENERAL_NAME_OTHER_NAME	The name is encoded as an OtherName type of GeneralName as specified in RFC 2459.
0x0001	X509_GENERAL_NAME_RFC822_NAME	The name is encoded as an IA5String type of GeneralName as specified in RFC 2459.
0x0002	X509_GENERAL_NAME_DNS_NAME	The name is encoded as an ORAddress type of GeneralName as specified in RFC 2459.
0x0003	X509_GENERAL_X400_ADDRESS	The name is encoded as an IA5String type of GeneralName as specified in RFC 2459.
0x0004	X509_GENERAL_NAME_DIRECTORY_NAME	The name is encoded as a Name type of GeneralName as specified in RFC 2459.
0x0005	X509_GENERAL_NAME_EDI_PARTY_NAME	The name is encoded as an EDIPartyName type of GeneralName as specified in RFC 2459
0x0006	X509_GENERAL_NAME_UNIFORM_RESOURCE_IDENTIFIER	The name is encoded as an IA5String type of GeneralName as specified in RFC 2459.
0x0007	X509_GENERAL_NAME_IP_ADDRESS	The name is encoded as n OCTECT STRING type of GeneralName in "network byte order" as specified by ASN.1, RFC 2459 and RFC 791.

Value	Name	Description
0x0008	X509_GENERAL_NAME_REGISTERED_ID	The name is encoded as an OBJECT IDENTIFIER type of GeneralName as specified in ASN.1. and RFC 2459.

4.4 Key Usage Extension Values

The X.509 key usage extension is used to specify for what purpose a key should be used. When an application goes through the verification process, it typically checks whether the key is being used for a purpose that it was not intended for.

The value of this extension is returned in a pointer to a *nuint16* by NPKIT_x509KeyUsageInfo (page 65) where each bit specifies a key usage. Any combination of key usages can be present but not all are appropriate combinations or are appropriate for all types of keys.

The following values for key usages are defined:

Value	Name	Description
0x8000	X509_KEY_USAGE_DIGITAL_SIGNATURE	For creating digital signatures.
0x4000	X509_KEY_USAGE_NON_REPUDIATION	For non-repudiation. This type of key usually has legal ramifications.
0x2000	X509_KEY_USAGE_KEY_ENCIPHERMENT	For encrypting other keys.
0x1000	X509_KEY_USAGE_DATA_ENCIPHERMENT	For directly encrypting data.
0x0800	X509_KEY_USAGE_KEY_AGREEMENT	For key agreement (for example, when a Diffie_Hellman key is to be used for key management). Not valid for RSA keys.
0x0400	X509_KEY_USAGE_KEY_CERT_SIGN	For signing certificates.
0x0200	X509_KEY_USAGE_CRL_SIGN	For signing CRLs.
0x0100	X509_KEY_USAGE_ENCIPHER_ONLY	Only for enciphering data while performing key agreement (X509_KEY_USAGE_KEY_AGREEMENT must also be set). Not valid for RSA keys
0x0080	X509_KEY_USAGE_DECIPHER_ONLY	Only for deciphering data while performing key agreement (X509_KEY_USAGE_KEY_AGREEMENT must also be set). Not valid for RSA keys.

4.5 NPKI_VerifyCallBackStruct Flag Values

One of the following defined values should be passed into the flags field of the function NPKIVerifyCertificateChain or in the flags field of the NPKI_VerifyCallBackStruct:

Value	Name	Description
0x00000000	NPKI_VERIFY_NORMAL	Verifies the certificate chain along with Certificate Revocation Lists (CRL).
0x00000001	NPKI_VERIFY_DONT_CHECK_CERTIFICATE	Do not verify the certificate chain.
0x00000002	NPKI_VERIFY_DONT_CHECK_CRL	Do not check CRLs.

4.6 NPKIT_Version Values

The NPKIT NDK Version number can be used to compare with the return value of NPKIT_Version (page 48) to determine if the version of the NPKIT library.

Value	Constant	Description
0x00010000	PKIS_VERSION_ONE_ZERO_ZERO	PKIS NDK version 1.0.
0x00010005	PKIS_VERSION_ONE_ZERO_FIVE	PKIS NDK version 1.0.5.
0x00010009	PKIS_VERSION_ONE_ZERO_NINE	PKIS NDK version 1.0.9.
0x00020000	PKIS_VERSION_TWO_ZERO_ZERO	PKIS NDK version 2.0.
0x00020002	PKIS_VERSION_TWO_ZERO_TWO	PKIS NDK version 2.0.2.
0x00020003	PKIS_VERSION_TWO_ZERO_THREE	PKIS NDK version 2.0.3.
0x00020011	PKIS_VERSION_TWO_ONE_ONE	PKIS NDK version 2.1.1.
0x00020200	PKIS_VERSION_TWO_TWO_ZERO	PKIS NDK version 2.2.
0x00020201	PKIS_VERSION_TWO_TWO_ONE	PKIS NDK version 2.2.1.
0x00020400	PKIS_VERSION_TWO_FOUR_ZERO	PKIS NDK version 2.4.
0x00020500	PKIS_VERSION_TWO_FIVE_ZERO	PKIS NDK version 2.5.
0x00020502	PKIS_VERSION_TWO_FIVE_TWO	PKIS NDK version 2.5.2.
0x00020504	PKIS_VERSION_TWO_FIVE_FOUR	PKIS NDK version 2.5.4.
0x00020600	PKIS_VERSION_TWO_SIX_ZERO	PKIS NDK version 2.6.
0x00020700	PKIS_VERSION_TWO_SEVEN_ZERO	PKIS NDK version 2.7.
0x00020702	PKIS_VERSION_TWO_SEVEN_TWO	PKIS NDK version 2.7.2.
0x00020703	PKIS_VERSION_TWO_SEVEN_THREE	PKIS NDK version 2.7.3.
0x00020704	PKIS_VERSION_TWO_SEVEN_FOUR	PKIS NDK version 2.7.4.
0x00020705	PKIS_VERSION_TWO_SEVEN_FIVE	PKIS NDK version 2.7.5.
0x00020706	PKIS_VERSION_TWO_SEVEN_SIX	PKIS NDK version 2.7.6.
0x00020707	PKIS_VERSION_TWO_SEVEN_SEVEN	PKIS NDK version 2.7.7.
0x00020708	PKIS_VERSION_TWO_SEVEN_EIGHT	PKIS NDK version 2.7.8.

Value	Constant	Description
0x00020709	PKIS_VERSION_TWO_SEVEN_NINE	PKIS NDK version 2.7.9.
0x00030000	PKIS_VERSION_THREE_ZERO_ZERO	PKIS NDK version 3.0.0.
0x00030100	PKIS_VERSION_THREE_ONE_ZERO	PKIS NDK version 3.1.0.
0x00030101	PKIS_VERSION_THREE_ONE_ONE	PKIS NDK version 3.1.1.
NPKIT_VERSION	PKIS_VERSION_THREE_ONE_ONE	The current NPKIT version 3.1.1.
NPKI_VERSION	PKIS_VERSION_THREE_ONE_ONE	The current NPKI version 3.1.1.
NPKI_CERTIFICATE_SERVER_VERSION	PKIS_VERSION_THREE_ONE_ONE	The current NPKI certificate server version 3.1.1.
NPKI_INSTALL_VERSION	PKIS_VERSION_THREE_ONE_ONE	The current NPKI install version 3.1.1.

4.7 NPKIT_x509 Certificate Invalidity Reason Flags

The following flags are used to specify why a certificate is invalid. These values are used by the *cRLReason* parameter in the functions NPKIT_VerifyCertChain (page 43), NPKIT_VerifyCertChainWithCallback (page 46), and NPKIT_VerifyCertChain (page 43).

Value	Name	Description
0x0000000	NPKIx509CertificateValid	The certificate is valid.
0x0000001	NPKIx509Invalid_System_Error	Hardware or network problems were encountered.
0x0000002	NPKIx509Invalid_Decode_Error	There was a problem decoding the certificate.
0x0000003	NPKIx509Invalid_Subject_Issuer_Name	The subject name of the issuing certificate does not match the issuer name of subject certificate.
0x0000004	NPKIx509Invalid_Future	The certificate's start date is in the future.
0x0000005	NPKIx509Invalid_Expired	The certificate has expired.
0x0000006	NPKIx509Invalid_Issuer_Not_CA	The issuer is not a valid CA.
0x0000007	NPKIx509Invalid_Path_Length	The X.509 basic constraints path length has been violated.
0x0000008	NPKIx509Invalid_Unknown_Critical_Extension	The certificate contains a critical extension that can not be understood.
0x0000009	NPKIx509Invalid_KeyUsage	The key does not support the requested usage.
0x000000A	NPKIx509Invalid_CRL_Decode_Error	An error occurred during the decoding of the CRL.

Value	Name	Description
0x000000B	NPKIx509Invalid_Certificate_On_CRL	One of the certificates in the chain is on a CRL.
0x000000C	NPKIx509Invalid_Cant_Process_CDP	The certificate contains a distribution point that can not be processed.
0x000000D	NPKIx509Invalid_Cant_Read_CRL	The CRL could not be read.
0x000000E	NPKIx509Invalid_Invalid_CRL	The CRL is not valid for this certificate.
0x000000F	NPKIx509Invalid_Expired_CRL	The CRL has expired and a new one has not been issued.
0x0000010	NPKIx509Invalid_CRL_Issuer_Name	The issuer name of the CRL identified in the certificate does not match the issuer name in the CRL retrieved.
0x0000011	NPKIx509Invalid_Issuer_Not_Trusted	One or more of the certificates in the certificate chain does not exist in the specified trusted root container. **NOTE:** This error code can only be returned by a call to NPKIVerifyCertificateWithTrustedRoots, and not any of the NPKIT functions.
0x0000012	NPKIx509Invalid_CDP_Exists_Did_Not_Check_CRL	(An advisory flag.) The CDP (Certificate Distribution Point) exists, but the CRL was not checked because you requested that it not be checked.
0x0000013	NPKIx509Invalid_Invalid_Signature	The signature of the CRL is invalid.

4.7.1 NPKIT_x509 CRL Distribution Point Reason Code

The following flags are used to specify why a CRL distribution point is invalid. These values are used by the *reasons* parameter in the function NPKIT_x509CRLDistributionPoint (page 51).

Value	Name	Description
0	PKI_UNSPECIFIED	The reason is not specified.
1	PKI_KEY_COMPROMISED	Invalid because the key was compromised.
2	PKI_CA_COMPROMISED	Invalid because the certificate authority was compromised.
3	PKI_AFFILIATION_CHANGED	Invalid because the certificate's affiliation was inappropriately changed.
4	PKI_SUPERSEDED	Invalid because the certificate has been superseded.
5	PKI_CESSATION_OF_OPERATION	Invalid because the CA is no longer operational.
6	PKI_CERTIFICATE_HOLD	Invalid because the certificate has been placed on hold.

Value	Name	Description
7	PKI_PRIVILEDGE_WITHDRAWN	Invalid because the user's privileges have been withdrawn by the CA.
8	PKI_AA_COMPROMISE	Invalid because the certificate has been compromised.

4.8 NPKIT_x509 CRL Types Values

One of the following values is returned in the *crlIssuerType* of NPKIT_x509CRLDistributionPoint (page 51) function.

Value	Name	Description
0x00000000	NPKIT_x509UnknownType	The CRL is of an unknown type.
0x00000001	NPKIT_x509CRLType	The CRL is a general type, containing all reason codes.
0x00000002	NPKIT_x509DeltaCRLType	The CRL is a delta CRL.
0x00000004	NPKIT_x509OnlyUserCertsType	The CRL contains only user certificates.
0x00000008	NPKIT_x509OnlyCACertsType	The CRL contains only CA certificates.
0x00000010	NPKIT_x509OnlySomeReasonsType	The CRL contains only certificates that were revoked for the specified reasons.
0x00000020	NPKIT_x509IndirectCRLType	The CRL is an indirect CRL.

4.9 NPKIT_x509 CRL Hold Types

The following flags are used to specify what to do if a certificate is on hold. These values are used by the *cRLHoldInstruction* parameter in the functions NPKIT_VerifyCertChain (page 43), NPKIT_VerifyCertChainWithCallback (page 46), and NPKIT_VerifyIssuerSubjectNameMatch (page 47).

Value	Name	Description
0	PKI_HOLD_INSTRUCTION_NONE	No hold instructions.
1	PKI_HOLD_INSTRUCTION_CALL_ISSUER	The person trying to verify the certificate should contact the certificate's issuer for information.
2	PKI_HOLD_INSTRUCTION_REJECT	The certificate should be rejected as if it were revoked.

4.10 X.509 Extensions

The extensions of a X.509 certificate provide a generic way to include information in the certificate. Currently the function provides support for including five X.509 extensions types: key usage, basic constraints, issuer alternative name, subject alternative name, and the Novell Security Attributes.

To provide a generic method of specifying data for X.509 extensions, the API provides general-purpose data structures and defines, as well as extension-specific data structures and defines.

The following lists extensions from enum NPKIT_x509Extension (see also NPKIT_CRLEntryExtensionInfo (page 32), NPKIT_CRLExtensionInfo (page 36), and NPKIT_x509GetExtensionData (page 60).

Extension Name	Description
None	No type has been specified.
Unknown	This extension that cannot be handled properly by NPKIT.
DecodeError	The extension type cannot be determined because of an error during decoding.
AuthorityKeyIdentifier	Provides a means of identifying the particular public key used to sign a certificate. This extension is used when an issuer has multiple signing keys (either because of multiple concurrent key pairs or changeover). Generally, this extension should be included in certificates.
SubjectKeyIdentifier	Provides a means of identifying the particular public key used in an application. When a reference to a public key identifier is needed (such as with the use an Authority Key Identifier) and is not included in the associated certificate, a SHA-1 hash of the subject public key is used. The hash is calculated over the value (excluding tag and length) of the subject public key field in the certificate.
KeyUsage	Defines the purpose (for example, encipherment, signature, certificate signing, etc.) of the key contained in the certificate.
PrivateKeyUsagePeriod	Allows the certificate issuer to specify a different validity period for the private key than the certificate. This extension is intended for use with digital signature keys and consists of two optional components: `notBefore` and `notAfter`. The private key associated with the certificate should not be used to sign objects before or after the times specified by the two components, respectively. CAs conforming to this profile should not generate certificates with private key usage period extensions unless at least one of the two components is present.
CertificatePolicies	Specifies the policy under which the certificate has been issued and the purposes for which the certificate can be used, defined by a sequence of policy information terms, each consisting of an Object Identifier (OID™) and optional qualifier.
PolicyMapping	Extension to be used within CA certificates for listing one or more pairs of object identifiers, each of them including an issuerDomainPolicy and a subjectDomainPolicy. The pairing indicates the issuing CA considers its issuerDomainPolicy equivalent to the subject CA's subjectDomainPolicy.
SubjectAltName	Allows you to bind additional identities to the subject of the certificate. Defined options include an RFC822 name (e-mail address), a DNS name, IP address, and URL.
IssuerAltName	Extension for associating Internet style identities with the issuer of the certificate. Defined options include an rfc822 name (electronic mail address), a DNS name, an IP address, and an URL.

Extension Name	Description
SubjectDirectoryAttributes	This extension is not recommended, but it can be used in local environments. **IMPORTANT:** This extension must be non-critical.
BasicConstraints	Identifies whether the subject of the certificate is a CA and how deep a certification path can exist through that CA.
NameConstraints	Specifies a name space within which all subject names in subsequent certificates in a certification path must be located. Restrictions might apply to the subject distinguished name or subject alternative names. Restrictions are defined in terms of permitted or excluded name subtrees. Any name matching a restriction in the `excludedSubtrees` field is invalid regardless of information appearing in the `permittedSubtrees`.
PolicyConstraints	Constrains path validation in two ways; it can be used to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier. The policy constraints extension can be used in certificates issued to CAs.
CRLDistributionPoints	Specifies how CRL information is obtained.
ExtendedKeyUsage	Specifies one or more purposes for which the certified public key can be used, in addition to or in place of the basic purposes indicated in the key usage extension.
AuthorityInfoAccess	Specifies how to access CA information and services for the issuer of the certificate in which the extension appears. Information and services can include on-line validation services and CA policy data. (The location of CRLs is not specified in this extension; that information is provided by the cRLDistributionPoints extension.) This extension can be included in subject or CA certificates, and it MUST be non-critical.
NovellAttribute	Specifies Novell Security Attributes. For full information, see NPKIT_x509NovellExtensionInfo (page 67).
CRLNumber	Conveys a monotonically increasing sequence number for each CRL issued by a given CA through a specific CA X.500 directory entry or CRL distribution point. This extension allows users to easily determine when a particular CRL supersedes another CRL.
ReasonCode	Identifies the reason for a certificate revocation. CAs are strongly encouraged to include reason codes in CRL entries; however, do not include a reason code CRL entry extension when using the unspecified reasonCode value.
InstructionCode	A non-critical CRL entry extension that provides a registered instruction identifier indicating the action to be taken after encountering a certificate that has been placed on hold.

Extension Name	Description
InvalidityDate	A non-critical CRL entry extension that provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date might be earlier than the revocation date in the CRL entry, which is the date at which the CA processed the revocation. When a revocation is first posted by a CA in a CRL, the invalidity date can precede the date of issue of earlier CRLs, but the revocation date must not precede the date of issue of earlier CRLs. Whenever this information is available, CAs are strongly encouraged to share it with CRL users.
DeltaCRLIndicator	A critical CRL extension that identifies a delta CRL. The use of delta-CRLs can significantly improve processing time for applications that store revocation information in a format other than the CRL structure. This allows changes to be added to the local database while ignoring unchanged information. **IMPORTANT:** When a delta CRL is issued, the CAs also must issue a complete CRL.
IssuingDistributionPoint	A critical CRL extension that identifies the CRL distribution point for a particular CRL and indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, or a limited set of reason codes. Although the extension is critical, conforming implementations are not required to support this extension.
CertificateIssuer	Identifies the certificate issuer associated with an entry in an indirect CRL (that is, a CRL that has the indirectCRL indicator set in its issuing distribution point extension). If this extension is not present on the first entry in an indirect CRL, the certificate issuer defaults to the CRL issuer. On subsequent entries in an indirect CRL, if this extension is not present, the certificate issuer for the entry is the same as that for the preceding entry.

4.11 Subject Alternative Name Extension Values (obsolete, 3/2005)

The X.509 subject alternative name extension were formerly used to specify additional identities to be bound to the subject of the certificate (that is, other names that identify the object) but are now deprecated. That functionality is now provided by Section 4.3, “General Name Type Extensions,” on page 74.

The subject alternative name type specified which encoding format was used to encode the alternative name. The following subject alternative name types were defined:

Value	Name	Description
0x0000	X509_SUBJECT_ALT_NAME_OTHER_NAME	An OtherName sequence as specified in RFC 2459.
0x0001	X509_SUBJECT_ALT_NAME_RFC822_NAME	Unicode representation of an IA5String.
0x0002	X509_SUBJECT_ALT_NAME_DNS_NAME	Unicode representation of an IA5String.

Value	Name	Description
0x0003	X509_SUBJECT_ALT_NAME_X400_ADDRESS	An ORAddress sequence as specified in RFC 2459.
0x0004	X509_SUBJECT_ALT_NAME_DIRECTORY_NAME	A **Name** choice as specified in x.501.
0x0005	X509_SUBJECT_ALT_NAME_EDI_PARTY_NAME	An `EDIPartyName` sequence as specified in RFC 2459.
0x0006	X509_SUBJECT_ALT_NAME_UNIFORM_RESOURCE_IDENTIFIER	A Unicode representation of an IA5String.
0x0007	X509_SUBJECT_ALT_NAME_IP_ADDRESS	An OCTET STRING in network byte order as specified in ASN.1. (Network byte order is specified in RFC 791.)
0x0008	X509_SUBJECT_ALT_NAME_REGISTERED_ID	An OBJECT IDENTIFIER as specified in ASN.1.

Structures

5

This section provides the structures used by NPKIT:

NPKI_CertChain

Contains information defining a certificate chain.

Structure

```
typedef struct NPKI_CertChain
{
     pnuint8                 cert;
     nuint32                 certLen;
     NPKI_CRL               *cRLStruct;
     nuint32                 flags;
     nuint32                 numErrors;
     NPKI_ERROR             *error;
     void                    reserved1;
     void                    reserved2;
     struct NPKI_CertChain  *next;
}NPKI_CertChain;
```

Fields

cert

Points to the certificate.

certLen

Specifies the length in bytes of the certificate.

cRLStruct

Optional, not supported yet. Pass in NULL.

flags

Future use, pass in NULL.

numErrors

Future use, set to zero.

error

Future use, set to NULL.

reserved1

Future use, set to NULL.

reserved2

Future use, set to NULL.

next

Points to the next NPKI_CertChain structure containing the next certificate in the chain.

Remarks

This structure is used to pass a certificate chain to NPKIT_VerifyCertChain (page 43) and is used in the `certificateChain` field of the structure Section 4.5, "NPKI_VerifyCallBackStruct Flag Values," on page 75". The certificates in the certificate chain need to be in order, leaf to root.

NPKI_VerifyCallBackStruct

Contains information defining a callback structure.

Structure

```
typedef struct NPKI_VerifyCallBackStruct
{
     void                  (*callback) (void *data);
     NPKI_CertChain         *certificateChain;
     nuint32                 flags;
     NWRCODE                 ccode;
     nuint32                 cRLReason;
     nuint32                 cRLHoldInstruction;
     time_t                  cRLRevocationTime;
     time_t                  cRLInvalidityDateTime;
     NPKI_CertChain         *revokedCertificate;
     nuint32                 certInvalidityReason;
     void                   *reserved1;
     void                   *reserved2;
     void                   *reserved3;
     void                   *userData;
}NPKI_VerifyCallBackStruct;
```

Fields

`callback (void* data)`

(IN) Points to the user-defined callback function.

`certificateChain`

(IN) Points to a linked list of NPKIT_VerifyCertChain (page 43) structures containing the certificate chain in leaf to root order.

`flags`

(IN) Pass in one of the following flags:

NPKI_VERIFY_NORMAL
NPKI_VERIFY_DONT_CHECK_CERTIFICATE
NPKI_VERIFY_DONT_CHECK_CRL

For more information, see Section 4.5, “NPKI_VerifyCallBackStruct Flag Values,” on page 75.

`ccode`

(OUT) Specifies whether the verify function succeeded or not.

`cRLReason`

(OUT) Specifies the reason code from the CRL (only when certificate is on the CRL). This field is set only when the ccode is set to PKI_E_CERT_INVALID.

`cRLHoldInstruction`

(OUT) Specifies the hold instruction from the CRL. This field is set only when the ccode is set to PKI_E_CERT_INVALID.

cRLRevocationTime

(OUT) Specifies the time of certificate revocation. This field is set only when the `ccode` is set to PKI_E_CERT_INVALID.

cRLInvalidityDateTime

(OUT) Specifies the time the certificate became invalid (only when certificate is on the CRL and *invalidityDate* is specified on the CRL). Thisfield is set only if the `ccode` is set to PKI_E_CERT_INVALID.

revokedCertificate

(OUT) Points to the invalid certificate. This field is set only when the `ccode` is set to PKI_E_CERT_INVALID.

certInvalidityReason

(OUT) Code to specify why the certificate is invalid. This will only be set if the `ccode` is set to PKI_E_CERT_INVALID.

reserved1

Reserved for future use.

reserved2

Reserved for future use.

reserved3

Reserved for future use.

userData

(IN) Points to a user-defined data structure. This field is available for you to pass information to the callback function.

Remarks

Points to the following structure is passed into the data field of the function NPKIT_VerifyCertChainWithCallback (page 46).

NOTE: This structure must be allocated and deallocated by the caller.

Revision History

A

Revision Date	Changes
October 11, 2006	Fixed broken links.

March 1, 2006	• Updated offsite links to provide more recent resource data.
	• Document the following NPKIT functions:
	  • NPKIT_CacheAddElement (page 14)
	  • NPKIT_CacheAddPKCS12Elements (page 16)
	  • NPKIT_CacheClearAllElements (page 18)
	  • NPKIT_CacheCreateContext (page 19)
	  • NPKIT_CacheElementInfo (page 20)
	  • NPKIT_CacheExportToPKCS12 (page 22)
	  • NPKIT_CacheFreeContext (page 24)
	  • NPKIT_CacheRead (page 25)
	  • NPKIT_CacheWrite (page 27)

October 5, 2005	• Transitioned to revised Novell documentation standards.
October 6, 2005	• Added Section 4.7.1, “NPKIT_x509 CRL Distribution Point Reason Code,” on page 78 for NPKIT_x509CRLDistributionPoint (page 51).
March 2, 2005	• Made technical corrections and fixed broken links. • Deprecated Section 4.11, “Subject Alternative Name Extension Values (obsolete, 3/2005),” on page 82 and replaced with the revised Section 4.3, “General Name Type Extensions,” on page 74. • Added the Section 4.7, “NPKIT_x509 Certificate Invalidity Reason Flags,” on page 77. • Added Section 4.6, “NPKIT_Version Values,” on page 76.
October 6, 2004	Made technical corrections and fixed broken links.
8 October 2003	Updated links to other Novell Certificate Server™ for C Libraries.
June 2003	Edited entire document.
March 2003	Fixed broken links in original documentation and added links to external sample code.
January 2003	Added as a new component to the Novell Certificate Server Libraries.