Novell Developer Kit

www.novell.com

October 11, 2006

NOVELL CERTIFICATE SERVER™ LIBRARY FOR C VERSION 2

Novell®

Legal Notices

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.

Contents

About This Guide

Novell® Certificate Server™ Library for C Version 2 (NPKI) furnishes you with a directory-centered, public key infrastructure (PKI) to create, 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 Windows* NT*/2000/XP, NetWare™ and AIX*.

For more specific information about this API and how it relates to other Novell Certificate Server APIs, see Novell Certificate Server APIs — Overview.

This guide contains the following sections:

Additional Information

For more comprehensive background information about setting up, managing, and troubleshooting this service, see the Novell Certificate Server Administration Guide (http://www.novell.com/documentation/lg/crt221ad/index.html).

The new Certificate Server functionality runs only on the same platforms as eDirectory 8.7 (see Novell eDirectory 8.7 System Requirements (http://www.novell.com/products/edirectory/sysreqs.html)

For Certificate Server source code projects, visit Forge Project: Novell Certificate Server Libraries for C (http://forge.novell.com/modules/xfmod/project/?ncslib) and Forge Project: Novell Certificate Server Classes for Java (http://forge.novell.com/modules/xfmod/project/?ncsjava).

For Certificate Server sample code, see Novell Forge Files Novell Certificate Server Libraries for C (../../../samplecode/ncslib_sample/index.htm).

For help with Certificate Server problems or questions, visit the Novell NCSLIB Support Forum (http://developer-forums.novell.com/group/novell.devsup.ncslib/readerNoFrame.tpt/@thread@first).

Documentation Updates

For the most recent version of this guide, see Novell Certificate Server Libraries for C (http://developer.novell.com/ndk/ncslib.htm).

Documentation Conventions

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

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

User Comments

We want to hear your comments and suggestions about this manual. To contact us, send e-mail to ndk@novell.com.

Getting Started

1

For more conceptual information about Novell® Certificate Server™ Library for C Version 2 (NPKI), see Concepts in the Novell Certificate Server APIs Overview document.

1.1 NPKI Dependencies

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

1.2 Getting Started

Before calling any of the Novell Certificate Server functions, you must create a context, set the tree name, and login and authenticate to the tree. Call these APIs in the following order:

1. NPKICreateContext (page 28)
2. NPKISetTreeName (page 106)
3. NPKIConnectToIPAddress (page 27)
4. NPKIDSLogin (page 51)

When these steps are completed, you need to finish up and clear the context by calling NPKIFreeContext (page 69) (see Section 2.23, "Housekeeping Tasks," on page 20).

For an example of how to use the login functions, see LoggingIn (../../../samplecode/ncslib_sample/LoggingIn.cpp.html).

1.3 Retrieving API Version Information

To obtain NPKIAPI version information, call NPKIVersionInfo (page 118). For a sample implementation of this task, see VersionInfo (../../../samplecode/ncslib_sample/VersionInfo.cpp.html).

1.4 Getting Server Information

When creating certificates, you need to determine what key sizes, algorithms, and validity dates are supported by the servers used in this process. Call NPKIGetServerInfo (page 88) to get this information from each of the servers. For a sample implementation of this task, see GetServerInfo (../../../samplecode/ncslib_sample/GetServerInfo.cpp.html).

Tasks 2

Novell® Certificate Server™ Library for C Version 2 contains functions that allow you to create, manage, and store user and server certificates. Examples of the tasks described in this section are listed in Chapter 6, “NPKI Sample Code,” on page 137.

This section contains the following tasks:

2.1 Creating a Certificate Authority

When creating a Certificate Authority (CA), you should choose a server that is highly available and highly reliable, then follow this procedure:

1. Determine that a CA does not already exist by calling NPKIFindOrganizationalCA (page 64), which returns an error if the CA does not exist.
2. Call NPKIGetServerUTCTime (page 94) to get the current time on the server that will host the CA.

3. Call NPKIGetServerInfo (page 88) and NPKIGetAlgorithmInfo (page 73) to determine the key sizes, algorithms, and validity dates that are supported on the server.
4. Determine the certificate attributes and extensions, then create the certificate by calling NPKICreateOrganizationalCA (page 29).

For an example implementation of this task, see CreateCA (../../../samplecode/ncslib_sample/CreateCA.cpp.html).

2.2 Retrieving CA Certificates

CA certificates are retrieved by calling NPKIGetCACertificates (page 75).

1. You can call NPKIFindOrganizationalCA (page 64) to get the Distinguished Name (DN) of the CA.
2. Call NPKIGetCACertificates (page 75) to retrieve the certificates. Most applications should only use the self-signed certificate.

For a sample implementation of this task, see GetCACert (../../../samplecode/ncslib_sample/GetCACert.cpp.html).

2.3 Backing Up the CA

The CA certificates and private key can be backed by calling NPKIExportCAKey (page 54) function.

1. Call NPKIFindOrganizationalCA (page 64) to get the DN of the CA. The CA name is the leaf name of the DN.
2. Call NPKIExportCAKey (page 54) to export the CA's private key and certificates into a personal information exchange (PFX) file. Store the PFX file in a secure place.

For a sample implementation of this task, see BackupCA (../../../samplecode/ncslib_sample/BackupCA.cpp.html).

2.4 Restoring the CA

To restore the certificate authority, call NPKIImportCAKey (page 98) to import the CA's private key and certificates into eDirectory™ using the PFX file you previously created using NPKIExportCAKey (page 54).

For a sample implementation of this task, see RestoreCA (../../../samplecode/ncslib_sample/RestoreCA.cpp.html).

2.5 Creating User Certificates

The first step in creating a user certificate is to find a CA and retrieve information from it. The second step is to find a key generation server and retrieve information from it. The third step is to determine the certificates attributes and extensions using the information from the previous two steps and user input.

1. Find the CA by calling NPKIFindOrganizationalCA (page 64). Then call these accessor functions:
   - NPKIGetHostServerDN (page 80)—to get the name of the server that hosts the CA
   - NPKIGetServerUTCTime (page 94)—to get the current time on the server which hosts the CA
   - NPKIGetServerInfo (page 88)—to get the supported key signature algorithms and the maximum and minimum validity times
2. Only servers holding a writeable partition that contains the user's object can create certificates for the user. Call NPKIFindKeyGenServersForUser (page 60) to find a server that holds this partition. After a successful return, you can use the following accessor functions:
   - NPKIServerNames (page 105)—to retrieve the servers' DN
   - NPKIGetServerInfo (page 88)—to get the supported key generation algorithm and to determine whether the key generation server is the same server as the CA server
   - NPKIGetAlgorithmInfo (page 73)—to get the maximum supported key generation sizes.
3. Determine the certificate attributes and extensions, then create the user certificate by calling NPKICreateUserCertificate (page 42).

For a sample implementation of this task, see CreateUserCert (../../../samplecode/ncslib_sample/CreateUserCert.cpp.html).

2.6 Importing a User Certificate

To import a certificate for a user, call NPKIStoreUserCertificate (page 112) to store the certificate on the user's object in eDirectory. The certificate is stored on the industry standard attribute *userCertificate*.

For a sample implementation of this task, see ImportUserCert (../../../samplecode/ncslib_sample/ImportUserCert.cpp.html).

2.7 Retrieving User Certificates

The first step in retrieving user certificates is to find the certificates based on search criteria. The second step is to so get each certificate and associated information using the accessor function provided.

1. To find user certificates, call NPKIFindUserCertificates (page 66). Other than the mandatory field of *userDN*, all other search criteria are optional. Only certificates that match all of your search criteria are returned.

   NOTE: : If the *nickName* field is used, all other search criteria is ignored.

2. After a successful return, call the accessor function NPKIUserCertInfo (page 114) to retrieve each certificate and its associated information.

For a sample implementation of this task, see FindUserCerts (../../../samplecode/ncslib_sample/FindUserCerts.cpp.html).

2.8 Reading a User's Private Key Nicknames

The first step in getting all of a user's private key nicknames is to read all of the names. The second step is to retrieve each of the names.

1. To read the private key nicknames for a user, call NPKIReadAllNickNames (page 103).
2. After a successful return, use the accessor function NPKINickName (page 102) to retrieve each of nicknames.

For a sample implementation of this task, see ReadUserNicknames (../../../samplecode/ncslib_sample/ReadUserNicknames.cpp.html).

2.9 Exporting a User's Private Key

To export a user's private key and certificate chain, call NPKIExportUserKey (page 58) to create a PFX file. Store the PFX file in a secure place.

For a sample implementation of this task, see ExportUserCerts (../../../samplecode/ncslib_sample/ExportUserCert.cpp.html).

2.10 Creating Server Certificates (Internal CA)

The first step in creating a server certificate is to find and retrieve information from the server for which you wish to create a certificate. The second step is to find the CA and retrieve information from it. The third step is to determine the certificates attributes and extensions using the information from the previous two steps and user input.

1. Find the server for which you want to create a certificate by calling NPKIFindKeyGenServersForUser (page 60). Then call these accessor functions:
   - NPKIServerNames (page 105)—to retrieve the servers' DN
   - NPKIGetServerInfo (page 88)—to get the supported key generation algorithm and to determine whether the key generation server is the same server as the CA server
   - NPKIGetAlgorithmInfo (page 73)—to get the maximum supported key generation sizes
2. Find the CA by calling NPKIFindOrganizationalCA (page 64). Then call these accessor functions:
   - NPKIGetHostServerDN (page 80)—to get the name of the server that hosts the CA
   - NPKIGetServerUTCTime (page 94)—to get the current time on the server that hosts the CA
   - NPKIGetServerInfo (page 88)—to get the supported key signature algorithms and the maximum and minimum validity times
3. Determine the certificate attributes and extensions, then create the server certificate by calling NPKICreateServerCertificate (page 34). The server certificate must be stored once it is created (see Section 2.11, “Storing Server Certificates (Internal CA),” on page 17).

   For a sample implementation of this task, see CreateServerCert (../../../samplecode/ncslib_sample/CreateServerCert.cpp.html).

   IMPORTANT: During creation of server certificates, if the key-generation server is the same as the CA server, you should not store the certificates.

2.11 Storing Server Certificates (Internal CA)

Server certificates should be added to the certificate list by calling NPKICertificateList (page 22), and then stored by a calling NPKIStoreServerCertificatesFromCertificateList (page 110).

IMPORTANT: During creation of server certificates, if the key-generation server is the same as the CA server, you should not store the certificates.

After a successful call to NPKICreateServerCertificate (page 34) in a multiserver environment, the certificates need to be stored as follows:

1. NPKICertificateList (page 22)—using the clear flag to delete all old certificates from the list
2. NPKICertInfo (page 24)—to get the server certificate that was just created by the successful call to NPKICreateServerCertificate (page 34)
3. NPKICertificateList (page 22)—using the add flag to add the server certificate to the list
4. NPKIGetCACertificates (page 75)—to get the CA's self-signed certificate
5. NPKICertificateList (page 22)—using the add flag ORed with the sort flag to add the CA's certificate and to sort the list
6. NPKIStoreServerCertificatesFromCertificateList (page 110)—to store the certificate list

For a sample implementation of this task, see CreateServerCert (../../../samplecode/ncslib_sample/CreateServerCert.cpp.html).

2.12 Creating Server Certificates (External CA)

There are two major tasks when creating an externally signed certificate. The first task is to create a PKCS #10 Certificate Signing Request (CSR). Follow the steps below to create the CSR.

Send the CSR to the external CA and retrieve the resulting certificate and all of the CA's certificates. Then store all of the certificates in the server certificate object (see Section 2.13, "Storing Server Certificates (External CA)," on page 18). For information about how to store the certificates,

1. Find the server for which you want to create a certificate by calling NPKIFindKeyGenServersForUser (page 60). Then call the following accessor functions:
   - NPKIServerNames (page 105)—to retrieve the servers' DN
   - NPKIGetServerInfo (page 88)—to get the supported key generation algorithm and to determine whether the key generation server is the same server as the CA server
   - NPKIGetAlgorithmInfo (page 73)—to get the maximum supported key generation sizes
2. Determine the certificate attributes and extensions, then create the server CSR (Certificate Signing Request) by calling NPKICreateServerCertificate (page 34). Send the CSR to the external CA to get the server certificate. The server certificate must be stored once it is created (see Section 2.13, "Storing Server Certificates (External CA)," on page 18).

For a sample implementation of this task, see GenerateCSR (../../../samplecode/ncslib_sample/GenerateCSR.cpp.html).

2.13 Storing Server Certificates (External CA)

Server certificates should be added to the certificate list by calling NPKICertificateList (page 22), then stored by a calling NPKIStoreServerCertificatesFromCertificateList (page 110).

1. NPKICertificateList (page 22)—using the clear flag to delete all old certificates from the list
2. NPKICertificateList (page 22)—call this function for each of the certificates in the certificate chain, using the add flag to add each certificate to the list
3. NPKICertificateList (page 22)—using the sort flag to sort the list
4. NPKIStoreServerCertificatesFromCertificateList (page 110)—to store the certificate list

For a sample implementation of this task, see StoreServerCerts (../../../samplecode/ncslib_sample/StoreServerCerts.cpp.html).

2.14 Retrieving Server Certificates

Server certificates are retrieved by calling NPKIGetServerCertificates (page 83). If the certificates' in the chain are needed, they can be accessed by calling NPKIChainCertInfo (page 25).

1. Call NPKIFindServerCertificateNames (page 62) to get a list of all certificate names for a server. For each certificate name, call NPKIGetServerCertificateStatus (page 85) to determine whether server certificates are available.
2. Call NPKIGetServerCertificates (page 83) to retrieve the server certificates and the number of certificates in the chain.
3. To access the chain certificates, call NPKIChainCertInfo (page 25).

For a sample implementation of this task, see RetrieveServerCertificate (../../../samplecode/ncslib_sample/RetrieveServerCertificate.cpp.html).

2.15 Backing Up a Server Certificate

To back up a server certificate, call NPKIExportServerKey (page 56) to export the server’s private key and certificates into a PFX file. Store the PFX file in a secure place.

For a sample implementation of this task, see BackupServerCertificate (../../../samplecode/ncslib_sample/BackupServerCertificate.cpp.html).

2.16 Restoring a Server Certificate

To restore the certificate authority call NPKIImportServerKey (page 100) to import the server’s private-key and certificates into eDirectory using the PFX file you previously created using NPKIExportServerKey (page 56).For a sample implementation of this task, see RestoreServerCertificate (../../../samplecode/ncslib_sample/RestoreServerCertificate.cpp.html).

2.17 Retrieving A Server's Private Key

There are two separate ways to retrieve a server’s private key. The first method is to retrieve a server’s private key securely wrapped in the server’s storage key. In this form, the key has been cryptographically protected from disclosure and can only be unwrapped and used by NICI running on the server.

The second way to retrieve a server's private key is to get a NICI handle to the key. The key can then be used by your NICI enabled application.

1. Call NPKIGetWrappedServerKey (page 97) to retrieve the server's private key securely wrapped in the server's storage key.
2. Call NPKIGetHandleToServerKey (page 77) to get a NICI handle to the server's private key.

For a sample implementation of this task, see GetServerKey (../../../samplecode/ncslib_sample/GetServerKey.cpp.html).

2.18 Creating a Certificate from a CSR

The first step in creating a certificate from a CSR is to find a CA and retrieve information from it. The second step is to determine the certificates attributes and extensions using the information from the previous step and user input.

1. Find the CA by calling NPKIFindOrganizationalCA (page 64). Then call these accessor functions:
   - NPKIGetHostServerDN (page 80)—to get the name of the server that hosts the CA
   - NPKIGetServerUTCTime (page 94)—to get the current time on the server which hosts the CA
   - NPKIGetServerInfo (page 88)—to get the supported key signature algorithms and the maximum and minimum validity times
2. Determine the certificate attributes and extensions, then create the certificate by calling NPKIGenerateCertificateFromCSR (page 70).

For a sample implementation of this task, see SignCSR (../../../samplecode/ncslib_sample/SignCSR.cpp.html).

2.19 Retrieving IP and DNS Information

You can retrieve IP and DNS addresses (as determined by WinSock) using the following functions:

1. Retrieve the IP and DNS information by calling NPKIGetServerIPAndDNSInfo (page 93), which returns the number of IP addresses.
2. To retrieve each of the IP addresses, call NPKIGetServerIPAddress (page 91) to return the IP address and the number of DNS names associated with that address.
3. To retrieve each of the DNS names associated with the previously retrieved IP address, call NPKIGetServerDNSName (page 87).

For a sample implementation of this task, see GetIPandDNSInfo (../../../samplecode/ncslib_sample/GetIPandDNSInfo.cpp.html).

2.20 Creating a Trusted Root Container

To create a trusted root container, call NPKICreateTrustedRootContainer (page 41). Trusted root containers can be created within any other container.See the sample implementation of this task in CreateTrustedRootContainer (../../../samplecode/ncslib_sample/CreateTrustedRootContainer.cpp.html).

2.21 Creating a Trusted Root Object

To add a trusted root to a trusted root container, call NPKICreateTrustedRoot (page 39). Trusted root objects can only be created in a trusted root container.

IMPORTANT: The trusted root container must already exist.

See the sample implementation of these tasks in the CreateTrustedRoot (../../../samplecode/ncslib_sample/CreateTrustedRoot.cpp.html).

2.22 Verifying Certificates with a Trusted Root

To properly verify a certificate, the entire certificate chain needs to be verified. The NPKIAPI libraries provide the capability to store CA certificates (that is, trusted roots) within eDirectory in a trusted root container. Use NPKIVerifyCertificateWithTrustedRoots (page 116) to have the NPKIAPI libraries attempt to construct the entire certificate chain, using certificates found in the specified trusted root container, and to verify the resulting chain.

For a sample implementation of this task, see VerifyWithTrustedRoot (../../../samplecode/ncslib_sample/VerifyWithTrustedRoot.cpp.html).

2.23 Housekeeping Tasks

When you are finished with all your operations, you need to logout and clean up the context you created. For an example of how to use the clean up, see LoggingIn (../../../samplecode/ncslib_sample/LoggingIn.cpp.html).

Functions 3

Novell® Certificate Server™ Library for C Version 2 furnishes you with a directory-centered, public key infrastructure to create, manage, and access X.509 certificates. The API is provided entirely in the C programming language to provide the best cross-platform support for all platforms integrating eDirectory.

NOTE: The *NDK: Novell Certificate Server Classes for Java* also are available.

This API is currently supported on all platforms supported by eDirectory 8.7.*x* For specific configuration and implementation requirements, see Novell eDirectory 8.7 System Requirements (http://www.novell.com/products/edirectory/sysreqs.html).

Novell Certificate Server interfaces, prototypes, and data types are defined in the Novell header files npki.h and nverify.h. As in all earlier versions, the API is subject to change.

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

NPKICertificateList

Stores a certificate (such as, X.509) or set of certificates (such as, PKCS #7) to an internal structure.

Syntax

```
#include "npki.h"

NWRCODE NPKICertificateList(
   const NPKIContext   context,
   const pnuint8       certificate,
   const nuint32       certificateLen,
   const nuint32       flags,
   pnuint32            numberOfCertsInList);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`certificate`

(IN) Specifies the X.509 certificate or PKCS #7 certificate set to be acted upon.

`certificateLen`

(IN) Specifies the length in bytes of `certificate`.

`flags`

(IN) Specifies the task to preform on the certificate being passed in. Use one or more of the following flags:

- PKI_ADD_CERT—Used when a certificate is being added to the certificate list. This flag can be used alone or with PKI_SORT_LIST. Using this flag with PKI_DEL_CERT causes an error.
- PKI_DEL_CERT—Used when a certificate is removed from the certificate list. The parameter certificate must point to a valid X.509 DER-encoded certificate. This flag can be used alone or with PKI_SORT_LIST. Using this flag with PKI_ADD_CERT causes an error
- PKI_CLEAR_CERTS—Used to delete all the certificates that have been stored with previous calls to NPKICertificateList. The certificate parameter should be NULL. This flag must be used alone.
- PKI_SORT_LIST—Used to sort this list of certificates. PKI_E_BROKEN_CHAIN is returned if the certificates do not form a complete chain. This flag can be used with either PKI_ADD_CERT or PKI_DEL_CERT. When used alone, certificate can be NULL.

`numberOfCertsInList`

(OUT) Specifies the number of certificates in the list.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

Each call to NPKICertificateList can store, remove, and/or sort the internal certificate chain structure. A subsequent call to NPKIStoreServerCertificatesFromCertificateList (page 110) stores the chain of certificates to a Key Material Object (KMO).

When creating a server certificate in a multi-server environment for a server that does not host the CA, calling NPKICreateServerCertificate (page 34) just creates the KMO (that is, it does not store the certificate or certificate chain). After a successful call to NPKICreateServerCertificate, call NPKICertInfo (page 24) to get the object certificate that was just created. Call NPKICertificateList (page 22) with the flag PKI_CLEAR_CERTS to make sure the internal certificate list is cleared.

Add the object certificate to the certificate list by calling NPKICertificateList and passing in the object certificate, object certificate length, and the flag PKI_ADD_CERT. Call NPKIGetCACertificates (page 75) to get the self-signed certificate. Add this to the certificate list. When all certificates have been added and a complete chain has been assembled, call NPKIStoreServerCertificatesFromCertificateList (page 110) to store the certificates in the list to the KMO.

See Also

NPKIStoreServerCertificatesFromCertificateList (page 110)

NPKICertInfo

Retrieves a newly created X.509 certificate with its corresponding size (formerly NWPKICertInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKICertInfo(
   const NPKIContext     context,
   pnuint32              certSize,
   nuint8 const        **cert);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

certSize

(OUT) Returns the size of the certificate.

cert

(OUT) Returns a constant pointer to the X.509 DER-encoded certificate.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

A successful call to NPKIGenerateCertificateFromCSR (page 70) or NPKICreateServerCertificate (page 34) must be made immediately before calling this function.

See Also

NPKICreateServerCertificate (page 34), NPKIGenerateCertificateFromCSR (page 70)

NPKIChainCertInfo

Obtains a pointer to the specified X.509 certificate in a certificate chain, and the size of the certificate (formerly NWPKIChainCertInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIChainCertInfo(
   const NPKIContext       context,
   const nuint32           index,
   pnuint32                certSize,
   nuint8 const          **cert,
   void                   *reserved1,
   void                   *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Indicates which certificate is to be returned.

NOTE: *index* is 0 based.

`certSize`

(OUT) Specifies the certificate size.

`cert`

(OUT) Points to the DER-encoded X.509 certificate.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

A successful call to either NPKIGetCACertificates (page 75) or NPKIGetServerCertificates (page 83) must have been made before calling this function.

See Also

NPKIGetCACertificates (page 75), NPKIGetServerCertificates (page 83)

NPKIConnectToIPAddress

Establishes a connection to the server at the specified IP address for the specified NPKIContext.

Syntax

```
#include "npki.h"

NWRCODE NPKIConnectToIPAddress(
   const NPKIContext    context,
   const nuint32        flags,
   const nuint16        port,
   const char          *iPAddress,
   const unicode       *treeName,
   const unicode       *serverDN);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`flags`

(IN) Reserved for future use; pass zero.

`port`

(IN) Specifies the port number to be used. If zero is passed in, the default IP port (524) is used.

`iPAddress`

(IN) Points to the IP address to use in the format *XXX.XXX.XXX.XXX*.

`treeName`

Returns the name of the tree that the server is in.

`serverDN`

Returns the fully distinguished name of the server.

Return Values

Returns 0 if successful, or a PKI or eDirectory error if not successful.

Remarks

You should call NPKIConnectToIPAddress after NPKISetTreeName (page 106) and before NPKIDSLogin (page 51).

NPKICreateContext

Creates a new PKI context structure and initializes it with default values (formerly NWPKICreateContext).

Syntax

```
#include "npki.h"

 NWRCODE NPKICreateContext(
    NPKIContext        *context);
```

Parameters

`context`

(OUT) Points to the newly created context handle.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

See Also

NPKIFreeContext (page 69)

NPKICreateOrganizationalCA

Creates the Organizational (that is, Tree) Certificate Authority (CA) if one does not already exist (formerly NWPKICreateOrganizationalCA).

Syntax

```
#include "npki.h"

NWRCODE NPKICreateOrganizationalCA(
   const NPKIContext            context,
   const unicode               *serverDN,
   const unicode               *organizationalCAName,
   const nuint32                keyType,
   const nuint32                keySize,
   const unicode               *subjectDN,
   const nuint32                signatureAlgorithm,
   const nuint32                dateFlags,
   const nuint32                validFrom,
   const nuint32                validTo,
   const nuint32                publicKeyFlags,
   const nuint32                privateKeyFlags,
   const NPKI_Extension        *keyUsage,
   const NPKI_Extension        *basicConstraints,
   const NPKI_ExtAltNames      *altNames,
   const NPKI_Extension        *NovellAttr,
   const NPKI_ASN1_Extensions  *extensions,
   unicode const              **organizationalCADN,
   const nuint32                retryFlag,
   void                        *reserved1,
   void                        *reserved2);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

serverDN

(IN) Specifies the eDirectory Server that will host the organizational CA. This must be a valid eDirectory server in *contextDN*.

organizationalCAName

(IN) Specifies the CA object name.

keyType

(IN) Specifies the type of key that you want generated. For this release, the only supported key type is RSA or a value of PKI_RSA_ALGORITHM. (See “Key Generation Algorithms Defines” on page 125).

keySize

(IN) Specifies the size of the key that the caller wants to generate. If the key size requested cannot be generated, the server returns an error and no key is generated. Calling NPKIGetServerInfo (page 88) with flags set to PKI_CA_INFO, NPKIGetAlgorithmInfo (page 73) obtains the supported key size on the server.

`subjectDN`

(IN) Specifies the *subjectDN*. This is the name to be encoded in the `subject` field in the X.509 certificate. The subject field identifies the entity associated with the public/private key pair. (For more information, see RFC 2459*, Section 4.1.2.6 (http://www.ietf.org/rfc/rfc2459.txt?number=2459).)

`signatureAlgorithm`

(IN) Specifies the algorithm to use to sign the certificate. You can call NPKIGetServerInfo (page 88) to determine which signature algorithms are supported.

`dateFlags`

(IN) Specifies whether dates have a two-digit or four-digit year. For this release, set this to DEFAULT_YEAR_ENCODING.

`validFrom`

(IN) Specifies the beginning of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or as 0xFFFFFFFF to represent the current time on the server.

`validTo`

(IN) Specifies the end of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or as 0xFFFFFFFF to represent the greatest validity period available on the server.

You can call NPKIGetServerInfo (page 88) to determine the validity period supported by the server.

`publicKeyFlags`

(IN) Specifies the public key options to use when creating the key pair. For this release, use the define PUBLIC_KEY_SINGLE_SERVER together with any optional public key flags.

`privateKeyFlags`

(IN) Specifies the private key options to use when creating the key pair. For this release, use the define PRIVATE_KEY.

NOTE: There currently is one "Optional Private Key Flag" on page 129 (PRIVATE_KEY _EXTRACTABLE). To use this optional flag, it must be ORed with the value PRIVATE_KEY to enable extraction of the CA's private key into a PKCS #12 file. (PKCS #12 is the standard format for extracting and importing keys). This flag must be used to enable backup of the CA's private key.

For a sample implementation of this task, see BackupCA (../../../samplecode/ncslib_sample/BackupCA.cpp.html).

When using the PRIVATE_KEY_EXTRACTABLE flag and including the Novell Security Attributes™ extension, it is necessary to bitwise-OR the extractable option (that is, NOVELL_EXTENSION_EXTRACTABLE_KEY in "Additional Flags" on page 122) along with the appropriate Novell attribute (see NOVELL_EXTENSION_ORGCA_DEFAULT

(0x00400) in "Mutually Exclusive Flags" on page 122) to the flags field in the Novell Security Attributes extension.

`keyUsage`

(IN) Specifies the X.509 key usage extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.4, "Key Usage Extension," on page 121. The key usage extension is not included in the certificate if this parameter is NULL.

`basicConstraints`

(IN) Specifies the X.509 basic constraints extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and the Section 4.1, "Basic Constraints Extension," on page 119. The basic constraints extension is not included in the certificate if this parameter is NULL.

`altNames`

(IN) Specifies the X.509 subject alternative name extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and the Section 5.3, "Subject Alternative Names Extension," on page 134 for more details. The subject alternative names extension is not included in the certificate if this parameter is NULL.

`NovellAttr`

(IN) Specifies the Novell Security Attributes extension. For more information, see the sections Section 4.16, "X.509 Extensions," on page 130 and Section 4.5, "Novell Security Attributes Extension," on page 122. If this parameter is NULL, the default Novell Security Attributes extension for a CA is included in the certificate.

`extensions`

(IN) Not implemented for CA certificates in this release. Pass in NULL.

`organizationalCADN`

(OUT) Returns the CA object's distinguished name (DN). The leaf name is supplied by the caller in the field *organizationalCAName* and the system concatenates it with the Security container's name to get the CA object's FDN.

`retryFlag`

(IN) Specifies whether the call is a retry. When NPKICreateOrganizationalCA (page 29) is called, a Certificate Authority object is created; however, eDirectory may take some time to replicate the object.

Because of the possibility of replication delay, subsequent calls to NPKICreateOrganizationalCA (page 29) might be necessary (for example, if previous calls fail due to replication delay); however, subsequent calls should be made with the *retryFlag* set to PKI_RETRY so that the system does not try to create a new CA object. The error code that is usually associated with a replication delay is ERR_NO_SUCH_ENTRY, -601.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x 2222 92 02 Install CA

Remarks

NPKICreateOrganizationalCA creates a Certificate Authority (CA) object in the Security container if one does not exist.

This function gives `serverDN` supervisor (S) rights to the All_Attributes ACL of the CA object, sets the NDSPKI:Organizational CA DN attribute of the Security container to be the distinguished name of the CA object, and gives [Public] read (R) rights to the NDSPKI:Organizational CA DN attribute of the Security container.

This function makes the Install CA NCP call to serverDN. This causes PKI services to generate an RSA key pair, create two X.509 certificates (one self-signed and one signed by the server's machine unique key), and store all of this information in the CA object.

To have the ability to backup the CA's private key, you must use the optional private key flag, PRIVATE_KEY_EXTRACTABLE.

For a sample implementation of how to back up the CA, see BackupCA (../../../samplecode/ncslib_sample/BackupCA.cpp.html).

See Also

NPKIFindOrganizationalCA (page 64), NPKIGetAlgorithmInfo (page 73), NPKIGetServerInfo (page 88), NPKIGetServerUTCTime (page 94)

NPKICreateSASServiceObject

Creates the Secure Authentication Services (SAS) service object for the specified server.

Syntax

```
#include "npki.h"

NWRCODE NPKICreateSASServiceObject(
   const NPKIContext    context,
   const unicode       *serverName,
   const unicode       *contextDN);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverName`

(IN) Specifies the name of the server for which to create SAS service object.

`contextDN`

(IN) Specifies the context of the server.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

NPKICreateSASServiceObject does not relink Key Material Objects (KMO) to a server, nor does it relink a server to its KMOs. This function simply allows new KMOs to be linked with the server specified in `serverName`.

NPKICreateServerCertificate

Creates a server key pair as well as the corresponding X.509 certificate (formerly NWPKICreateServerCertificate).

Syntax

```
#include "npki.h"

NWRCODE NPKICreateServerCertificate(
   const NPKIContext              context,
   const unicode                 *keyGenServerDN,
   const unicode                 *signServerDN,
   const unicode                 *certificateName,
   const nuint32                  keyType,
   const nuint32                  keySize,
   const unicode                 *subjectDN,
   const nuint32                  signatureAlgorithm,
   const nuint32                  dateFlags,
   const nuint32                  validFrom,
   const nuint32                  validTo,
   const nuint32                  publicKeyFlags,
   const nuint32                  privateKeyFlags,
   const NPKI_Extension          *keyUsage,
   const NPKI_Extension          *basicConstraints,
   const NPKI_ExtAltNames        *altNames,
   const NPKI_Extension          *NovellAttr,
   const NPKI_ASN1_Extensions    *extensions,
   void                          *reserved1,
   void                          *reserved2);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

keyGenServerDN

(IN) Points to the eDirectory FDN of the server for which to generate the X.509 certificate.

signServerDN

(IN) Specifies the eDirectory FDN of the server that hosts the CA that will be used to generate the X.509 certificate.

certificateName

(IN) Specifies the certificate name to be used to identify the key pair and corresponding certificate.

keyType

(IN) Specifies the type of key that is to be generated. For this release, the only supported key type is RSA or a value of PKI_RSA_ALGORITHM (see "Key Generation Algorithms Defines" on page 125).

keySize

(IN) Specifies the requested size of the key to be generated. If the key size requested could not be generated, the server returns an error and no key is generated. Calling NPKIGetServerInfo (page 88) with flags set to PKI_SERVER_INFO, then NPKIGetAlgorithmInfo (page 73) obtains the supported key sizes on the server. The intersection of the key sizes and algorithms supported by the `keyGenServerDN` and the `signServerDN` are the valid key sizes and algorithms.

subjectDN

(IN) Specifies the `subjectDN`. This is the name to be encoded in the *subject* name field in the X.509 certificate. The subject field identifies the entity associated with the public/private key pair. (For more information, see RFC 2459*, Section 4.1.2.6 (http://www.ietf.org/rfc/rfc2459.txt?number=2459).)

signatureAlgorithm

(IN) Specifies the signature algorithm to use to sign the certificates. You can call NPKIGetServerInfo (page 88) to determine which signature algorithms are supported.

dateFlags

(IN) Specifies whether dates have a two-digit or four-digit year. For this release, set this to DEFAULT_YEAR_ENCODING.

validFrom

(IN) Specifies the beginning of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the current time on the server.

validTo

(IN) Specifies the end of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the greatest validity period available on the server. You can call NPKIGetServerInfo (page 88) to determine the greatest validity period supported by the server.

publicKeyFlags

(IN) Specifies the public key options to use when creating the key pair. Use one of the following flags, together with any optional public key flags:

- PUBLIC_KEY_SINGLE_SERVER—Used when the signing server is the same as the key generation server. This is only possible when the key generation server also hosts a CA.
- PUBLIC_KEY_TWO_SERVER—Used when the signing server is not the same as the key generation server.
- PUBLIC_KEY_EXTERNAL_CA—Used when an external CA will sign the certificate.

privateKeyFlags

(IN) Specifies the private key options to use when creating the key pair. For this release use the define PRIVATE_KEY together with any optional private key flags.

NOTE: There currently is one "Optional Private Key Flag" on page 129 (PRIVATE_KEY _EXTRACTABLE). To use this optional flag, it must be OR'ed with the value PRIVATE_KEY to enable extraction of the server's private key into PKCS #12 file (PKCS #12 is the standard format for extracting and importing keys). This flag must be used to enable backup of the server's private key.

When using the PRIVATE_KEY_EXTRACTABLE flag and including the Novell Security Attributes extension, it's necessary to bitwise-OR the extractable option (that is, NOVELL_EXTENSION_EXTRACTABLE_KEY) along with the appropriate Novell attribute to the flags field in the Novell Security Attributes extension (see "Mutually Exclusive Flags" on page 122).

`keyUsage`

(IN) Specifies the X.509 key usage extension. For more information, see the Section 4.16, "X.509 Extensions," on page 130 and the Section 4.4, "Key Usage Extension," on page 121. The key usage extension is not included in the certificate if this parameter is NULL.

`basicConstraints`

(IN) Specifies the X.509 basic constraints extension. For more information, see the Section 4.16, "X.509 Extensions," on page 130 and the Section 4.1, "Basic Constraints Extension," on page 119. The basic constraints extension is not included in the certificate if this parameter is NULL.

`altNames`

(IN) Specifies the X.509 subject alternative name extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 5.3, "Subject Alternative Names Extension," on page 134. The subject alternative names extension is not included in the certificate if this parameter is NULL.

`NovellAttr`

(IN) Specifies the Novell Security Attributes extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.5, "Novell Security Attributes Extension," on page 122. If this parameter is NULL, the default Novell Security Attributes extension for a server certificate is included in the certificate.

`extensions`

(IN) Specifies any generic ASN.1 encoded extensions to add to the certificate. See Section 4.16, "X.509 Extensions," on page 130.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x 2222 93 03 Create Key Pair

0x 2222 93 04 Sign Certificate

Remarks

When calling NPKICreateServerCertificate (page 34), three different modes can be used: *single server mode*, *dual server mode*, or *external mode*. Depending on the mode selected, different NCPs™ are sent and different results occur.

The *single server mode* is used to create a server certificate when the signing server is the same as the key generation server. In this case, `signServerDN` should be set to NULL, and `publicKeyFlags` should consist of the define PUBLIC_KEY_SINGLE_SERVER combined with any optional public key flags desired.

After calling NPKICreateServerCertificate successfully, the newly generated server certificate and its corresponding certificate chain are stored in eDirectory. The newly generated server certificate is returned and can be accessed by calling NPKICertInfo (page 24).

NOTE: *Single server mode* is possible only when the key generation server also hosts a CA.

The *dual server mode* is used to generate a server certificate when the signing server is not the same as the key generation server. In this case `publicKeyFlags` should consist of the define PUBLIC_KEY_TWO_SERVER combined with any optional public key flags desired. The newly generated server certificate is returned and can be accessed by calling NPKICertInfo (page 24).

After calling NPKICreateServerCertificate successfully, it is necessary to store the newly generated certificate and its corresponding certificate chain.

You can retrieve the certificate chain by calling NPKIGetCACertificates (page 75) with the `flags` field set to PKI_OBJECT_KEY_CERT combined with PKI_SELF_SIGNED_CERT.

After the successful call to NPKIGetCACertificates (page 75), you should call NPKICertificateList (page 22) to add the certificates one at a time with the `flags` field set to PKI_ADD. Once all the certificates in the chain have been added, make the call again with the *flags* field set to PKI_SORT. You must call NPKIStoreServerCertificatesFromCertificateList (page 110) to actually store the certificates into the object.

The *external server mode* is used to generate a server certificate when an external CA signs the certificate. In this case, `signServerDN` is set to NULL and `publicKeyFlags` consists of the define PUBLIC_KEY_EXTERNAL_CA combined with any other public key flags desired. A PKCS #10 Public Key Signing Request (CSR) is generated and can be accessed by calling NPKICSRInfo (page 47).

The *external server mode* is used to generate a server Certificate Signing Request (CSR) to facilitate an external CA signing (or creating) the server certificate. For the *external server mode*, set `signServerDN` to NULL and `publicKeyFlags` to the define PUBLIC_KEY_EXTERNAL_CA combined with any optional public key flags desired. A PKCS #10 CSR is generated and accessed by calling NPKIStoreServerCertificates (page 107).

The CSR should be sent to the external CA. The external CA will send a new X.509 server certificate in response. The new X.509 server certificate signed (created) by the external CA, as well as the external CA's certificate chain, should be added by making calls to NPKICertificateList (page 22) with the `flags` field set to PKI_ADD. Once all the certificates in the chain have been added, make the call again with the `flags` field set to PKI_SORT. You must call NPKIStoreServerCertificatesFromCertificateList (page 110) to actually store the certificates into the object. This method of storing certificates will handle PKCS #7 files that contain multiple certificates

To have the ability to backup the server private key, the optional private key flag (PRIVATE_KEY_EXTRACTABLE) must be used. For a sample implementation of this task, see CreateServerCertificate. (../../../samplecode/ncslib_sample/CreateServerCert.cpp.html). For a sample implementation of how to back up the server certificate, see BackupServerCertificate (../../../samplecode/ncslib_sample/BackupServerCertificate.cpp.html).

See Also

NPKICertificateList (page 22)
NPKICertInfo (page 24)
NPKICSRInfo (page 47)
NPKIFindServerCertificateNames (page 62)
NPKIGetCACertificates (page 75)
NPKIGetServerCertificateStatus (page 85)
NPKIGetServerInfo (page 88)
NPKIServerCertificateName (page 104)
NPKIStoreServerCertificates (page 107)
NPKIStoreServerCertificatesFromCertificateList (page 110)

NPKICreateTrustedRoot

Creates a Trusted Root object and stores the specified X.509 root (or CA) certificate in the eDirectory object (formerly NWPKICreateTrustedRoot).

Syntax

```
#include "npki.h"

NWRCODE NPKICreateTrustedRoot(
   const NPKIContext     context,
   const unicode        *objectDN,
   const pnuint8         certificate,
   const nuint32         certificateLen);
```

Parameters

context

(IN) Specifies the NPKI `context` for the request.

objectDN

(IN) Specifies the eDirectory FDN of the Trusted Root object to be created. Trusted Root objects can be created only within Trusted Root containers.

certificate

(IN) Specifies the DER-encoded X.509 root (or CA) certificate you want to store in the Trusted Root object.

certificateLen

(IN) Specifies the size of the certificate.

Return Values

Returns 0 if successful or an eDirectory error code if not successful.

Remarks

Trusted Root containers along with Trusted Root objects provide a method of logically grouping, managing, and accessing X.509 root (CA) certificates within a directory service.

Trusted Root Objects can be created only within Trusted Root containers. Use NPKICreateTrustedRootContainer (page 41) to create a Trusted Root container.

NOTE: For a sample implementation of this task, see CreateTrustedRoot (../../../samplecode/ncslib_sample/CreateTrustedRoot.cpp.html).

See Also

NPKICreateTrustedRootContainer (page 41), NPKIFindTrustedRootsInContext (page 65), NPKIGetTrustedRootInfo (page 95)

NPKICreateTrustedRootContainer

Creates a container where Trusted Root objects can be created (formerly NWPKICreateTrustedRootContainer).

Syntax

```
#include "npki.h"

NWRCODE NPKICreateTrustedRootContainer(
   const NPKIContext          context,
   const unicode             *objectDN
);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

objectDN

(IN) Specifies the eDirectory FDN of the Trusted Root container to be created.

Return Values

Returns 0 if successful or an eDirectory error code if not successful.

Remarks

Trusted Root containers, along with Trusted Root objects, provide a method of logically grouping, managing, and accessing X.509 root (or CA) certificates within a directory service.

NOTE: For a sample implementation of this task, see CreateTrustedRootContainer (../../../samplecode/ncslib_sample/CreateTrustedRootContainer.cpp.html).

See Also

NPKICreateTrustedRoot (page 39), NPKIFindTrustedRootsInContext (page 65), NPKIGetTrustedRootInfo (page 95)

NPKICreateUserCertificate

Generates a key pair with its corresponding X.509 certificate (formerly NWPKICreateUserCertificate).

Syntax

```
#include "npki.h"

NWRCODE NPKICreateUserCertificate(
   const NPKIContext             context,
   const unicode                *keyGenServerDN,
   const unicode                *signServerDN,
   const unicode                *userDN,
   const unicode                *nickName,
   const nuint32                 keyType,
   const nuint32                 keySize,
   const unicode                *subjectDN,
   const nuint32                 signatureAlgorithm,
   const nuint32                 dateFlags,
   const nuint32                 validFrom,
   const nuint32                 validTo,
   const nuint32                 publicKeyFlags,
   const nuint32                 privateKeyFlags,
   const NPKI_Extension         *keyUsage,
   const NPKI_Extension         *basicConstraints,
   const NPKI_ExtAltNames       *altNames,
   const NPKI_Extension         *NovellAttr,
   const NPKI_ASN1_Extensions   *extensions,
   void                         *reserved1,
   void                         *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`keyGenServerDN`

(IN) Specifies the eDirectory FDN of the server used to generate the user's key pair.

`signServerDN`

(IN) Specifies the eDirectory FDN of the server that hosts the certificate authority that generates the X.509 certificate.

`userDN`

(IN) Specifies the FDN of the user object for which a certificate will be generated. This must be a valid eDirectory user object in the current tree.

`nickName`

(IN) Specifies the certificate nickname. This name must be unique for the specified user.

`keyType`

(IN) Specifies the type of key that you want to be generated. Call NPKIGetServerInfo (page 88) to get the supported key generation algorithms. For this release, the only supported key type is PKI_RSA_ALGORITHM (see "Key Generation Algorithms Defines" on page 125).

`keySize`

(IN) Specifies the requested size of the key to be generated. If the key size requested cannot be generated, an error is returned by the server and no key is generated. Calling NPKIGetServerInfo (page 88), then NPKIGetAlgorithmInfo (page 73) obtains the supported key sizes supported on the server. The intersection of the key sizes and algorithms supported by the *keyGenServerDN* and the *signServerDN* are the valid key sizes and algorithms.

`subjectDN`

(IN) Specifies the `subjectDN`. This is the name to be encoded in the subject field in the X.509 certificate. The subject field identifies the entity associated with the public/private key pair. (For more information, see RFC 2459*, Section 4.1.2.6 (http://www.ietf.org/rfc/rfc2459.txt?number=2459).)

This parameter should be NULL if the subject name (in the user certificate) is to be the user's typed FDN. If a name other than the eDirectory username is desired, this parameter must contain the typed FDN (and `publicKeyFlags` must include the flag PKI_CUSTOM_SUBJECT_NAME).

`signatureAlgorithm`

(IN) Specifies the signature algorithm to use to sign the certificate. To get the supported algorithms, call NPKIGetServerInfo (page 88). For this release, `signatureAlgorithm` must be set to one of the following:

- PKI_SIGN_WITH_RSA_AND_MD2
- PKI_SIGN_WITH_RSA_AND_MD5
- PKI_SIGN_WITH_RSA_AND_SHA1

`dateFlags`

(IN) Specifies whether dates have either a two-digit or four-digit year. For this release, set this to DEFAULT_YEAR_ENCODING.

`validFrom`

(IN) Specifies the beginning of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the current time on the server.

`validTo`

(IN) Specifies the end of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the greatest validity period available on the server.

You can call NPKIGetServerInfo (page 88) to determine the greatest validity period available on the server.

`publicKeyFlags`

(IN) Specifies the public key options to use when creating the key pair. Use one of the following flags, together with any optional public key flags:

- PUBLIC_KEY_SINGLE_SERVER—Used when the signing server is the same as the key generation server. This is possible only when the key generation server also hosts a CA.
- PUBLIC_KEY_TWO_SERVER—Used when the signing server is not the same as the key generation server.

`privateKeyFlags`

(IN) Specifies the private key options to use when creating the key pair. For this release use the define PRIVATE_KEY together with any optional private key flags.

NOTE: There currently is one "Optional Private Key Flag" on page 129 (PRIVATE_KEY _EXTRACTABLE). To use this optional flag, it must be bitwise-OR'ed with the value PRIVATE_KEY to enable extraction of a user's private key into a PKCS #12 file (PKCS #12 is the standard format to import keys into a browser).

When using the PRIVATE_KEY_EXTRACTABLE flag and including the Section 4.5, "Novell Security Attributes Extension," on page 122, it's necessary to bitwise-OR the extractable option (that is, NOVELL_EXTENSION_EXTRACTABLE_KEY) along with the appropriate Novell attribute (see NOVELL_EXTENSION_USER_DEFAULT in "Mutually Exclusive Flags" on page 122).

`keyUsage`

(IN) Specifies the X.509 key usage extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.4, "Key Usage Extension," on page 121. The key usage extension is not included in the certificate if NULL is passed in this parameter.

`basicConstraints`

(IN) Specifies the X.509 basic constraints extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.1, "Basic Constraints Extension," on page 119. The basic constraints extension is not included in the certificate if this parameter is NULL.

`altNames`

(IN) Specifies the X.509 subject alternative name extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 5.3, "Subject Alternative Names Extension," on page 134. The subject alternative names extension is not included in the certificate if this parameter is NULL.

`NovellAttr`

(IN) Specifies the Novell Security Attributes extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and the Section 4.5, "Novell Security Attributes Extension," on page 122. If this parameter is NULL, the default Novell Security Attributes for a user certificate is included in the certificate.

`extensions`

(IN) Specifies any generic ASN.1 encoded extensions to add to the certificate. For more information, see the Section 4.16, "X.509 Extensions," on page 130.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x2222 93 03 Create Key Pair
0x2222 93 04 Sign Certificate

Remarks

The key pair is stored securely in eDirectory as an attribute in the user's object. The private key is cryptographically wrapped using Novell International Cryptographic Infrastructure (NICI) to protect the key.

When calling NPKICreateUserCertificate (page 42), three different modes can be used: *single server mode*, *dual server mode*, or *external mode* (external mode is not supported in this release). Depending on the mode selected, different NCPs are sent and different results occur.

Single server mode is used to generate a user certificate when the signing server is the same as the key generation server. In this case, `signServerDN` is set to NULL, and `publicKeyFlags` consists of the define PUBLIC_KEY_SINGLE_SERVER combined with any optional public key flags desired. The newly generated user certificate is returned and can be accessed calling NPKIUserCertInfo (page 114).

NOTE: *Single server mode* is possible only when the key generation server also hosts a CA.

The *dual server mode* is used to generate a user certificate when the signing server is not the same as the key generation server. In this case `publicKeyFlags` consists of the define PUBLIC_KEY_TWO_SERVER combined with any optional public key flags desired. The newly generated user certificate is returned and can be accessed calling NPKIUserCertInfo (page 114).

If the error PKI_E_ADD_CERTIFICATE is returned when using *dual server mode*; although the certificate was created, it could not be stored in eDirectory because of replication delays. If this error occurs, you should call NPKIStoreUserCertificate (page 112) successfully. (The `userDN`, `nickName`, and `signServerDN` parameters should be the same as in NPKICreateUserCertificate, the flags parameter should be set to PKI_INTERNAL_KEY_PAIR, and all other parameters should be either NULL or 0.)

NOTE: The *external server mode* is not supported in this release.

The `signServerDN` must host a CA in the current tree. You can call NPKIFindKeyGenServersForUser (page 60) and NPKIGetServerInfo (page 88) to determine which servers meet the requirements to act as a CA for a specified user.

After a successful call to NPKICreateUserCertificate, the certificate and its length can be obtained by calling NPKIUserCertInfo using 0 in the *index* parameter.

See Also

NPKIDeleteUserCertificate (page 49), NPKIFindUserCertificates (page 66), NPKIStoreUserCertificate (page 112), NPKIUserCertInfo (page 114)

NPKICSRInfo

Obtains a pointer to a PKCS #10 Certificate Signing Request (CSR) and its corresponding size (formerly NWPKICSRInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKICSRInfo(
   const NPKIContext     context,
   pnuint32              csrSize,
   nuint8 const        **csr);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

csrSize

(OUT) Returns the size of the CSR.

csr

(OUT) Returns a pointer to a DER-encoded CSR.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

You must call NPKICreateServerCertificate (page 34) successfully (using the external CA method) immediately before calling this function.

See Also

NPKICreateServerCertificate (page 34)

NPKIDeleteDSObject

Deletes an eDirectory object (formerly NWPKIDeleteDSObject).

Syntax

```
#include "npki.h"

NWRCODE NPKIDeleteDSObject(
   const NPKIContext    context,
   const unicode       *objectDN);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

objectDN

(IN) Specifies the FDN of the eDirectory object to be deleted.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

Typically, this function is used only to clean up objects created during a failed create certificate operation.

WARNING: Caution should be used when calling this function because any object can be deleted if the caller has sufficient rights.

See Also

NPKICreateServerCertificate (page 34), NPKICreateOrganizationalCA (page 29)

NPKIDeleteUserCertificate

Deletes a user’s certificate (formerly NWPKIDeleteUserCertificate).

Syntax

```
#include "npki.h"

NWRCODE NPKIDeleteUserCertificate(
   const NPKIContext    context,
   const unicode       *userDN,
   const unicode       *nickName,
   const nuint32        flags,
   const pnuit8         certificate,
   const nuit32         certificateLength);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`userDN`

(IN) Specifies the FDN of a user object. This must be a valid eDirectory user object in the current tree.

`nickName`

(IN) Specifies the certificate nickname. This name is used to identify the key pair and associated certificate. It must be a valid certificate nickname for the specified user.

`flags`

(IN) This should currently be set to 0.

`certificate`

(IN) Specifies the DER-encoded X.509 certificate you want to delete.

`certificateLength`

(IN) Specifies the size of the certificate.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

The `certificate` and `certificateLength` variables are used if there is no nickname for the user certificate. This can happen if the certificate was added through LDAP. If the nickname field has a value, `certificate` and `certificateLength` can be `NULL`.

WARNING: Deleting a certificate can have severe consequences such as the inability to read encrypted email or encrypted files.

See Also

NPKICreateUserCertificate (page 42), NPKIFindUserCertificates (page 66)

NPKIDSLogin

Performs all authentication operations needed to establish a client’s connection to a network (formerly NWPKIDSLogin).

Syntax

```
include "npki.h"

 NWRCODE NPKIDSLogin(
    const NPKIContext     context,
    const unicode        *objectDN,
    const pnstr8          password);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`objectDN`

(IN) Specifies the FDN name of the object logging in to the network.

`password`

(IN) Specifies the object’s password.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

See Also

NPKIDSLogout (page 52)

NPKIDSLogout

Terminates an object’s connection to the network (formerly NWPKIDSLogout).

Syntax

```
include "npki.h"

 NWRCODE NPKIDSLogout(
    const NPKIContext     context);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

See Also

NPKIDSLogin (page 51), NPKISetTreeName (page 106)

NPKIDSObjectExists

Determines whether an eDirectory object exists.

Syntax

```
include "npki.h"

 NWRCODE NPKIDSObjectExists(
    const NPKIContext       context,
    const unicode           *objectDN);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`objectDN`

(IN) Points to the FDN of the object to be checked.

Return Values

Returns 0 if the object exists, or an eDirectory or PKI error code if not successful.

NPKIExportCAKey

Exports the CA's private key and corresponding certificates in Personal Information Exchange Syntax (PFX) format (formerly NWPKIExportCAKey).

Syntax

```
#include "npki.h"

void NPKIExportCAKey(
   NPKIContext const              context,
   const unicode                 *organizationlCAName,
   const unicode                 *password,
   const nuint32                  flags,
   pnuint32 const                 pfxSize,
   nuint8 const                 **pfx);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`organizationlCAName`

(IN) Specifies the FDN of the CA object (for example, if your CA is called Organizational CA and it exists in the Security container, this parameter should be set to Organizational CA). This must be a valid eDirectory name of a CA object in the current tree.

`password`

(IN) Specifies the password with which to encrypt the private key and certificate.

`flags`

(IN) Specifies options for exporting the server key and certificate. The flags currently defined are:PKI_CA_KEY_AND_CERTS—Exports the CA self-signed certificate and the chain of certificates in the certification (see Section 4.8, “NPKIExportCAKey Flags,” on page 123).

`pfxSize`

(OUT) Specifies the size of the exported data PFX.

`pfx`

(OUT) Points to the PKCS #12 encoded data.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x2222 93 09 Read Key

Remarks

The private key and certificates are encrypted using the input password as specified in the Public Key Cryptography Standards (PKCS) #12.

See Also

NPKIImportCAKey (page 98)

NPKIExportServerKey

Exports a server’s private key and corresponding certificates in Personal Information Exchange Syntax (PFX) format (formerly NWPKIExportServerKey).

Syntax

```
#include "npki.h"

void NPKIExportServerKey(
   const NPKIContext              context,
   const unicode                 *serverDN,
   const unicode                 *certificateName,
   const unicode                 *password,
   const nuint32                  flags,
   pnuint32 const                 pfxSize,
   nuint8 const                 **pfx);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN of the eDirectory server whose private key and certificates you want to export. This must be a valid eDirectory server in the current tree.

`certificateName`

(IN) Specifies which private key and certificates you want to export. It must be a valid certificate name for the specified server.

`password`

(IN) Specifies the password to use to encrypt the private key and certificate.

`flags`

(IN) Specifies options for exporting the server key and certificates (see Section 4.9, “NPKIExportServerKey Flags,” on page 124). The flags currently defined are:PKI_CHAIN_CERTIFICATE—Exports the chain of certificates in the certification path along with the specified server certificate.

`pfxSize`

(OUT) Specifies the size of the exported data PFX.

`pfx`

(OUT) Points to the PKCS #12 encoded data.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x2222 93 09 Read Key

Remarks

The key and certificate are encrypted using the input password as specified in the Public Key Cryptography Standards (PKCS) #12.

See Also

NPKIImportServerKey (page 100)

NPKIExportUserKey

Exports a private key and the corresponding certificates for the currently logged-in user in Personal Information Exchange Syntax (PFX) format (formerly NWPKIExportUserKey).

Syntax

```
#include "npki.h"

NWRCODE NPKIExportUserKey(
   const NPKIContext    context,
   const unicode       *nickname,
   const unicode       *password,
   const nuint32        flags,
   pnuint32            *pfxSize,
   nuint8 const        **pfx);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

nickname

(IN) Specifies the certificate nickname that identifies which private key and certificates are to be exported. `nickname` must be a valid certificate nickname for the currently logged-in user in the current tree.

password

(IN) Specifies the password to use to encrypt the private key and certificate.

flags

(IN) Specifies options for exporting the user key and certificates. The flags currently defined are:PKI_CHAIN_CERTIFICATE—Exports the chain of certificates in the certification path along with the specified user certificate.

pfxSize

(OUT) Points to the size of the exported data.

pfx

(OUT) Points to the PKCS #12 encoded data.

Return Values

Returns 0 if successful, or an eDirectory, NICI or PKI error code if not successful.

PKI NCP Calls

0x2222 93 09 Read Key

Remarks

The key and certificate are encrypted using the input password as specified in the PKCS #12. For a sample implementation of this task, see ExportUserCert (../../../samplecode/ncslib_sample/ExportUserCert.cpp.html).

NPKIFindKeyGenServersForUser

Finds the servers that can be used to generate a public/private key pair (that is, a certificate) for the users that reside in the specified name context (formerly NWPKIFindKeyGenServersForUser).

Syntax

```
#include "npki.h"

NWRCODE NWPKFindKeyGenServersForUser(
   const NPKIContext    context,
   const unicode       *nameContextDN,
   pnuint32              numberOfServers);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`nameContextDN`

(IN) Specifies the FDN context of the users for which you want to find a key generation server. This must be a valid container in the current tree.

`numberOfServers`

(OUT) Returns the number of servers with a read/write or master replica of the partition in which the user object resides.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

The requirements for a server to generate a key pair for a user are:

- The server must hold a read/write or master replica of the partition in which the user object resides.
- The server must be running the correct version of the Novell Certificate Server.

NPKIFindKeyGenServersForUser (page 60) finds all of servers that meet the first requirement. Call NPKIGetServerInfo (page 88) to determine if the selected server meets the second requirement. When a successful call to NPKIFindKeyGenServersForUser is made, the server names are stored in context specific values that can be accessed by calling NPKIServerNames (page 105).

NOTE: Multiple calls to NPKIServerNames are necessary if more than one server name is available.

See Also

NPKIServerNames (page 105)

NPKIFindServerCertificateNames

Finds the server certificate names for the specified server (formerly NWPKIFindServerCertificateNames).

Syntax

```
#include "npki.h"

NWRCODE NPKIFindServerCertificateNames(
   const NPKIContext    context,
   const unicode       *serverDN,
   pnuint32              numberOfCertificateNames);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the eDirectory FDN of the server.

`numberOfCertificateNames`

(OUT) Specifies the number of server certificate names available.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

Calls NPKIServerCertificateName (page 104) to retrieve the server certificate names.

See Also

NPKICreateServerCertificate (page 34), NPKIServerCertificateName (page 104)

NPKIFindServersInContext

Finds all of the NCP servers in the name context supplied (formerly NWPKIFindServersInContext).

Syntax

```
#include "npki.h"

NWRCODE NPKIFindServersInContext(
   const NPKIContext     context,
   const unicode        *nameContextDN,
   pnuint32              numberOfServers);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`nameContextDN`

(IN) Specifies the eDirectory FDN for which you want to find an NCP server. This must be a valid container in the current tree.

`numberOfServers`

(OUT) Specifies the number of NCP servers in `contextDN`.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

After a successful call, the server names are stored as non-typed relative names (that is, leaf or short names) in data values that can be accessed by calling NPKIServerNames (page 105).

See Also

NPKIServerNames (page 105)

NPKIFindOrganizationalCA

Finds and returns the name of the CA for the current tree (formerly NWPKIFindOrganizationalCA).

Syntax

```
#include "npki.h"

NWRCODE NPKIFindOrganizationalCA(
   const NPKIContext    context,
   unicode const      **objectDN);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`objectDN`

(OUT) Points to a Unicode* string that contains the name of either the organizational CA or the security container.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if unsuccessful. If the function returns 0, `objectDN` contains the name of the organizational CA. If the routine returns an error code of PKI_E_NO_TREE_CA, `objectDN` contains the security container's name. Any other error causes `objectDN` to be invalid.

Remarks

If CA cannot be found, this function returns the name of the security container.

See Also

NPKICreateOrganizationalCA (page 29), NPKISetTreeName (page 106)

NPKIFindTrustedRootsInContext

Finds all of the Trusted Root objects within the specified Trusted Root container and returns the number found (formerly NWPKIFindTrustedRootsInContext).

Syntax

```
#include "npki.h"

NWRCODE NPKIFindTrustedRootsInContext(
   const NPKIContext        context,
   const unicode           *nameContextDN,
   pnuint32                 numberOfTrustedRoots);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`nameContextDN`

(IN) Points to the eDirectory FDN of the Trusted Root container that is to be searched.

`numberOfTrustedRoots`

(OUT) Specifies the number of trusted roots in the trusted root container specified by `nameContextDN`.

Return Values

Returns 0 if successful or an eDirectory error code if not successful.

Remarks

For each root found, a call to NPKIGetTrustedRootInfo (page 95) can be made to retrieve the relevant information about the root.

Trusted Root containers along with Trusted Root objects provide a method of logically grouping, managing, and accessing X.509 root (or CA) certificates within a directory service.

See Also

NPKICreateTrustedRoot (page 39), NPKICreateTrustedRootContainer (page 41), NPKIGetTrustedRootInfo (page 95)

NPKIFindUserCertificates

Finds all of the certificates for the *userDN* that meets the search criteria, stores the certificates in context specific values, and returns the number of certificates that meet the search criteria (formerly NWPKIFindUserCertificates).

Syntax

```
#include "npki.h"

NWRCODE NPKIFindUserCertificates(
   const NPKIContext   context,
   const unicode       *userDN,
   const unicode       *nickName,
   const pnuint8        serialNumber,
   const nuint32        serialNumberLen,
   const nuint32        keyType,
   const nuint32        minKeySize,
   const nuint32        maxKeySize,
   const nuint32        searchOnKeyUsage,
   const nuint16        keyUsageValue,
   const unicode       *issuerDN,
   const unicode       *subjectDN,
   const nuint32        certificateValid,
   const nuint32        vendorID,
   const nuint32        certificateStatus,
   void                *reserved1,
   void                *reserved2,
   void                *reserved3,
   void                *reserved4,
   nuint32             *numberOfUserCerts);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

userDN

(IN) Specifies the FDN of the user for which you want to find a certificate. `userDN` must be a valid user object in the current tree.

nickName

(IN) (Optional) Specifies the certificate nickname that identifies which user certificate is to be read. `nickName` must be either NULL or a valid certificate nickname for the specified user.

serialNumber

(IN) (Optional) Specifies the certificate serial number. `serialNumber` must be either NULL or the serial number of a certificate for the specified user. If `serialNumber` is specified, `serialNumber` also must be specified.

serialNumberLen

(IN) (Optional) Specifies the length of the field `serialNumber` parameter. If `serialNumber` is specified, `serialNumber` must be specified. If you don't specify `serialNumber`, set serialNumber to zero.

`keyType`

(IN) (Optional) Specifies the algorithm type used to generate the public/private key pair. Currently the only algorithm supported is RSA (PKI_RSA_ALGORITHM) (see "Key Generation Algorithms Defines" on page 125). If you don't specify `keyType`, set it to zero.

`minKeySize`

(IN) (Optional) Specifies the minimum key size of the public/private key pair. If you don't specify `minKeySize`. set it to zero.

`maxKeySize`

(IN) (Optional) Specifies the maximum key size of the public/private key pair. If you don't specify `maxKeySize`, set it to zero.

`searchOnKeyUsage`

(IN) (Optional) Specifies whether to search using the `keyUsageValue` parameter. This parameter is necessary because a value of zero is valid for the `keyUsageValue` parameter. Set `searchOnKeyUsage` to TRUE or FALSE.

`keyUsageValue`

(IN) (Optional) Specifies the X.509 certificate extension, Key Usage. `keyUsage` is a bit field, and can either be zero (that is, not present or not specified) or it can be constructed using any valid combination of the following defines:

X509_KEY_USAGE_DIGITAL_SIGNATURE
X509_KEY_USAGE_NON_REPUDIATION
X509_KEY_USAGE_KEY_ENCIPHERMENT
X509_KEY_USAGE_DATA_ENCIPHERMENT
X509_KEY_USAGE_KEY_AGREEMENT
X509_KEY_USAGE_KEY_CERT_SIGN
X509_KEY_USAGE_CRL_SIGN
X509_KEY_USAGE_ENCIPHER_ONLY
X509_KEY_USAGE_DECIPHER_ONLY

`issuerDN`

(IN) (Optional) Specifies the X.509 FDN typed of the CA that issued the certificate. If you don't specify `issuerDN`, set it to NULL.

`subjectDN`

(IN) (Optional) Specifies the X.509 typed FDN of the subject of the certificate. If you don't specify `subjectDN`, set it to NULL.

`certificateValid`

(IN) (Optional) Specifies a specific date on which the requested certificate is valid. The date is represented as the number of seconds since 00:00:00 UTC January 1, 1970. If you don't specify `certificateValid`, set it to zero.

`vendorID`

(IN) (Optional) Specifies the vendor that issued the certificate. This parameter can be used to narrow the search to certificates supplied by a specific vendor. If you don't specify `vendorID`, set it to zero or PKI_ALL_VENDORS.

`certificateStatus`

(IN) (Optional) Specifies the status of the certificates you want to find. This parameter can be used to narrow the search to certificates that have a specific status. If you don't specify `certificateStatus`, set it to zero.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

`reserved3`

Reserved for future use.

`reserved4`

Reserved for future use.

`numberOfUserCerts`

(OUT) Returns the number of user certificates that meet the specified search criteria.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

Call NPKIUserCertInfo (page 114) to access the certificates and their sizes.

If you specify `nickName`, the certificate matching the nickname is returned (assuming a valid nickname) and all other search parameters are ignored. For all other cases, the set of certificates match all of the search criteria. If no search criteria are specified, all certificates for the user are available.

For sample code, see FindUserCerts (../../../samplecode/ncslib_sample/FindUserCerts.cpp.html).

See Also

NPKICreateUserCertificate (page 42), NPKIStoreUserCertificate (page 112), NPKIUserCertInfo (page 114)

NPKIFreeContext

Frees a previously allocated NPKI context and all associated memory.

Syntax

```
include "npki.h"

void NPKIFreeContext(
   NPKIContext     context);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

See Also

NPKICreateContext (page 28)

NPKIGenerateCertificateFromCSR

Accepts a PKCS #10 CSR from an external source and sends the request to `caServerDN`, which then creates and returns an X.509 certificate (formerly NWPKIGenerateCertificateFromCSR).

Syntax

```
#include "npki.h"

nuint32 NPKIGenerateCertificateFromCSR(
   const NPKIContext              context,
   const unicode                 *caServerDN,
   const pnuint8                  extCSR,
   const nuint32                  extCSRSize,
   const unicode                 *subjectDN,
   const nuint32                  signatureAlgorithm,
   const nuint32                  dateFlags,
   const nuint32                  validFrom,
   const nuint32                  validTo,
   const NPKI_Extension          *keyUsage,
   const NPKI_Extension          *basicConstraints,
   const NPKI_ExtAltNames        *altNames,
   const NPKI_Extension          *NovellAttr,
   const NPKI_ASN1_Extensions    *extensions,
   void                          *reserved1,
   void                          *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`caServerDN`

(IN) Specifies the FDN of the server that hosts the CA. This must be a valid eDirectory server in the current tree.

`extCSR`

(IN) Specifies the PKCS #10 CSR that is to be sent to the CA to create the X.509 certificate.

`extCSRSize`

(IN) Specifies the size of the PKCS #10 CSR.

`subjectDN`

(IN) Not supported in this release. Points to a subject name to use in the certificate, rather than using the subject name in the CSR. At this time, this parameter is ignored regardless of the value given.

`signatureAlgorithm`

(IN) Not supported in this release. Specifies the signature algorithm to use to sign the certificate, rather than using the signature algorithm in the CSR. Calls NPKIGetServerInfo (page 88) to determine which signature algorithms are supported. At this time, this parameter is ignored regardless of the value given.

`dateFlags`

(IN) Specifies whether dates have a two-digit or four-digit year. For this release, set to DEFAULT_YEAR_ENCODING.

`validFrom`

(IN) Specifies the beginning of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the current time on the server.

`validTo`

(IN) Specifies the end of the period of validity, represented as the number of seconds since 00:00:00 UTC Jan 1, 1970, or 0xFFFFFFFF to represent the greatest validity period available on the server. You call NPKIGetServerInfo (page 88) to determine the greatest validity period available on the server.

`keyUsage`

(IN) Specifies the X.509 key usage extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.4, "Key Usage Extension," on page 121. The key usage extension is not included in the certificate if this parameter is NULL.

`basicConstraints`

(IN) Specifies the X.509 basic constraints extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 4.1, "Basic Constraints Extension," on page 119. The basic constraints extension is not included in the certificate if this parameter is NULL.

`altNames`

(IN) Specifies the X.509 subject alternative name extension. For more information, see Section 4.16, "X.509 Extensions," on page 130 and Section 5.3, "Subject Alternative Names Extension," on page 134. The subject alternative names extension is not included in the certificate if this parameter is NULL.

`NovellAttr`

(IN) Specifies the Novell Security Attributes extension. For more information, see the Section 4.16, "X.509 Extensions," on page 130 and Section 4.5, "Novell Security Attributes Extension," on page 122. If this parameter is NULL, the default Novell Security Attributes extension for a key pair is created outside of the system.

`extensions`

(IN) Specifies any generic ASN.1 encoded extensions you want to add to the certificate. For more information, see Section 4.16, "X.509 Extensions," on page 130 and "General Purpose Extension Structure" on page 134.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x2222 93 04 Sign Certificate

Remarks

After a successful call, the resulting certificate and its size can be obtained by calling NPKICertInfo (page 24). The newly-created certificate is not stored in eDirectory.

See Also

NPKICertInfo (page 24), NPKIFindOrganizationalCA (page 64), NPKIGetServerInfo (page 88)

NPKIGetAlgorithmInfo

Obtains the supported key sizes for the specified algorithm (formerly NWPKIGetAlgorithmInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetAlgorithmInfo(
   const NPKIContext     context,
   const nuint32         algorithm,
   pnuint32              maxKeyEncryptKeySize,
   pnuint32              maxSigningKeySize,
   pnuint32              maxDataEncryptKeySize,
   void                 *reserved1,
   void                 *reserved2);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

algorithm

(IN) Specifies a bit mask that represents which algorithm information to return. The correct algorithms to use are the key-generation algorithms (as opposed to the signing algorithms) returned by NPKIGetServerInfo (page 88).

maxKeyEncryptKeySize

(OUT) Returns the maximum key size supported for use as a key encrypting key.

maxSigningKeySize

(OUT) Specifies the maximum key size supported for use as a key signing key.

maxDataEncryptKeySize

(OUT) Specifies the maximum key size supported for use as a data encrypting key.

reserved1

Reserved for future use.

reserved2

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

A successful call to NPKIGetServerInfo (page 88) must have been made immediately before calling this function.

See Also

NPKIGetServerInfo (page 88)

NPKIGetCACertificates

Reads the CA certificates for *objectDN* and stores them in context specific values (formerly NWPKIGetCACertificates).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetCACertificates(
   const NPKIContext    context,
   const unicode       *objectDN,
   const nuint32        flags,
   pnuint32             objectCertSize,
   nuint8 const       **objectCert,
   pnuint32             selfSignedCertSize,
   nuint8 const       **selfSignedCert,
   pnuint32             numberOfChainCerts,
   pnuint32             rootCertIndex,
   void                *reserved1,
   void                *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`objectDN`

(IN) Specifies the FDN of the object whose CA certificates you want. `objectDN` must be a valid CA object in the current tree.

`flags`

(IN) Specifies which certificates are read and stored. The following flags are currently defined:

- PKI_CHAIN_CERTIFICATE—Retrieves the certificate chain (that is, the chain rooted in the Novell Certifier CA). Only software that natively understands and processes the Novell Security Attributes Extension should use this chain.
- PKI_TRUSTED_ROOT_CERTIFICATE—Retrieves the trusted root certificate. Only software that natively understands and processes the Novell Security Attributes Extension should use this certificate.
- PKI_SELF_SIGNED_CERTIFICATE—Retrieves the self-signed certificate.

NOTE: Most applications should use this certificate.

- PKI_OBJECT_KEY_CERTIFICATE—Retrieves the object certificate (that is, the certificate for the specified object). Only software that natively understands and processes the Novell Security Attributes Extension should use this certificate.

PKI_OBJECT_KEY_CERTIFICATE can be combined with any of the other flags, but none of the other flags can be used together at one time. Also, NPKIGetServerCertificates (page 83) and NPKIGetCACertificates (page 75) use the same internal variables to store results, so calling either of these functions destroys the result of the previous call.

`objectCertSize`

(OUT) Returns the size of the object certificate.

`objectCert`

(OUT) Returns to the DER encoder X.509 object certificate.

`selfSignedCertSize`

(OUT) Returns the size of the self-signed certificate.

`selfSignedCert`

(OUT) Returns to the DER-encoded X.509 self-signed certificate.

`numberOfChainCerts`

(OUT) Returns the number of certificates in the certificate chain. You can call NPKIChainCertInfo (page 25) to retrieve the certificates in the certificate chain.

`rootCertIndex`

(OUT) Returns which certificate in the certificate chain is marked as the root certificate.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, PKI, or NetWare® error code if not successful.

PKI NCP Calls

0x2222 93 05 PKI Get Certificate

Remarks

The flags field determines which certificates are read. For sample code, see GetCACert (../../../samplecode/ncslib_sample/GetCACert.cpp.html).

See Also

NPKIChainCertInfo (page 25), NPKIFindOrganizationalCA (page 64)

NPKIGetHandleToServerKey

Obtains a NICI object handle to a server private key (formerly NWPKIGetHandleToServerKey).

Syntax

```
#include "npkikey.h"

NWRCODE NPKIGetHandleToServerKey(
   const NPKIContext    context,
   const unicode       *serverDN,
   const unicode       *certificateName,
   NICI_CC_HANDLE       ccsCtx,
   NICI_OBJECT_HANDLE  *pkiKeyHandle);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Points to the DN of the server object.

`certificateName`

(IN) Points to the certificate nickname that identifies which key is to be referenced. `certificateName` must be a valid server certificate name the specified server.

`ccsCtx`

(IN) Specifies the NICI context you must create and destroy this context. This context must be created and destroyed by the caller.

`pkiKeyHandle`

(OUT) Points to the handle to the NICI object for the private key specified by `certificateName`.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCPCalls

0x2222 93 09 Read Key

Remarks

After a successful return from this function, `pkiKeyHandle` contains a handle to a NICI object that can then be used in calling other NICI functions.

IMPORTANT: This function does not work unless it is called on the server for which the key was generated (that is, `serverDN`).

See Also

NPKIFindServerCertificateNames (page 62), NPKIServerCertificateName (page 104)

NPKIGetHandleToUserKey

Returns a NICI Object handle to a user key, specified by `nickname` (formerly NWPKIGetHandleToUserKey).

Syntax

```
#include "npkikey.h"

NWRCODE NPKIGetHandleToUserKey(
   const NPKIContext      context,
   const unicode         *nickname,
   NICI_CC_HANDLE         ccsCtx,
   NICI_OBJECT_HANDLE    *pkiKeyHandle);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`nickname`

(IN) Points to the certificate nickname that identifies which key to be referenced. `nickname` must be a valid certificate nickname for the currently logged-in user in the current tree. The corresponding private key must be stored in eDirectory.

`ccsCtx`

(IN) Specifies the NICI context. You must create and destroy this context.

`pkiKeyHandle`

(OUT) Points to the handle to the NICI object for the private key specified by the nickname.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCPCalls

0x2222 93 09 Read Key

Remarks

After a successful call the `pkiKeyHandle` contains a NICI object handle to the private key that can then be passed into other NICI calls to encrypt, decrypt or verify data.

See Also

NPKIFindUserCertificates (page 66), NPKIReadAllNickNames (page 103)

NPKIGetHostServerDN

Gets the DN of the server that is associated with the specified object (that is, reads the eDirectory attribute A_HOST_SERVER of `objectDN`) (formerly NWPKIGetHostServerDN).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetHostServerDN(
   const NPKIContext    context,
   const unicode       *objectDN,
   unicode const      **serverDN);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`objectDN`

(IN) Specifies the FDN of the object to read. This must be a valid eDirectory object name.

`serverDN`

(OUT) Returns the host server's FDN. This is a valid eDirectory object name.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

The eDirectory A_HOST_SERVER attribute is used on PKI and Secure Authentication Services (SAS) objects to identify which server hosts the object (or service).

See Also

NPKICreateServerCertificate (page 34), NPKIFindOrganizationalCA (page 64), NPKIGetServerInfo (page 88), NPKIUserCertInfo (page 114)

NPKIGetKMOCertificateName

Gets the certificate name for a key material object (KMO).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetKMOCertificateName
(
   const NPKIContext    context,
   const unicode       *kmoDN,
   unicode const      **certificateName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`kmoDN`

(IN) Points to the DN of the key material object.

`certificateName`

(OUT) Points to the name of the certificate.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

NPKIGetSASServiceName

Obtains the Secure Authentication Service (SAS) service name of the specified server object (formerly NWPKIGetSASServiceName).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetSASServiceName(
   const NPKIContext    context,
   const unicode       *serverDN,
   unicode const      **serviceName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the eDirectory server. This must be a valid eDirectory server in the current tree that has SAS installed.

`serviceName`

(OUT) Returns the SAS service name.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

NPKIGetServerCertificates

Obtains the certificates in a specified certificate set on a given server and stores them in context-specific values (formerly NWPKIGetServerCertificates).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerCertificates(
   const NPKIContext     context,
   const unicode        *serverDN,
   const unicode        *certificateName,
   const nuint32         flags,
   pnuint32              objectCertSize,
   nuint8 const        **objectCert,
   pnuint32              numberOfChainCerts,
   pnuint32              rootCertIndex,
   void                 *reserved1,
   void                 *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Points to the FDN of the eDirectory server whose certificates you want to obtain. This must be a valid eDirectory server in the current tree.

`certificateName`

(IN) Identifies which server certificate set you want to get.

`flags`

(IN) The `flags` field determines which certificates are read and stored. The following `flags` are currently defined:

- PKI_CHAIN_CERTIFICATE—Retrieves the certificate chain.
- PKI_TRUSTED_ROOT_CERTIFICATE—Retrieves the trusted root certificate.
- PKI_OBJECT_KEY_CERTIFICATE—Retrieves the object certificate (that is, the certificate for the specified object).
- PKI_CHAIN_CERTIFICATE and PKI_TRUSTED_ROOT_CERTIFICATE cannot be combined.

`objectCertSize`

(OUT) Specifies the size of the object certificate.

`objectCert`

(OUT) Points to the DER encoder X.509 object certificate.

`numberOfChainCerts`

(OUT) Specifies the number of certificates in the certificate chain. You can call NPKIChainCertInfo (page 25) to retrieve the certificates in the certificate chain.

`rootCertIndex`

(OUT) Specifies which certificate in the certificate chain is marked as the root certificate.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, PKI, or NetWare error code if not successful.

Remarks

The `flags` field determines which certificates are read. NPKIGetServerCertificates (page 83) and NPKIGetCACertificates (page 75) use the same internal variables to store results, so calling one function right after calling the other causes data to be overwritten.

PKI NCP Calls

0x2222 93 05 PKI Get Certificate.

See Also

NPKIChainCertInfo (page 25), NPKICreateServerCertificate (page 34), NPKIFindServerCertificateNames (page 62), NPKIGetCACertificates (page 75), NPKIServerCertificateName (page 104), NPKIStoreServerCertificates (page 107), NPKIStoreServerCertificatesFromCertificateList (page 110)

NPKIGetServerCertificateStatus

Determines the status of the server certificate (formerly NWPKIGetServerCertificateStatus).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerCertificateStatus(
   const NPKIContext     context,
   const unicode        *serverDN,
   const unicode        *certificateName,
   pnuint32              flags);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

serverDN

(IN) Specifies the FDN of the eDirectory server for which you want the status of a certificate. This must be a valid eDirectory server in the current tree.

certificateName

(IN) Specifies the certificate name that you want to get information about. This must be a valid certificate for the specified server.

flags

(OUT) Specifies the status of the server certificate.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

The server certificate can be in one of five states:

- KMO_EMPTY—The server certificate does not have a valid public-private key pair.
- KMO_KEY_PAIR_PRESENT—The server certificate has a valid public-private key pair.
- KMO_TRUSTED_ROOT_PRESENT—The server certificate has a valid public-private key pair and a valid trusted root.
- KMO_CERTIFICATE_PRESENT—The server certificate has a valid public-private key pair, a valid trusted root, and a valid object certificate.
- KMO_INVALID_STATE—The server certificate is in an invalid state.

See Also

NPKICreateServerCertificate (page 34), NPKIFindServerCertificateNames (page 62), NPKIGetCACertificates (page 75), NPKIServerCertificateName (page 104), NPKIStoreServerCertificates (page 107)

NPKIGetServerDNSName

Retrieves the DNS name for a server (formerly NWPKIGetServerDNSName).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerDNSName(
   const NPKIContext       context,
   nuint32                 index,
   unicode const         **DNSName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Specifies which DNS Name is to be returned. This DNS name is associated with the IP address returned by the previous successful call to NPKIGetServerIPAddress (page 91). `index` is 0 based.

`DNSName`

(OUT) Returns the specified DNS name.

Return Values

Returns 0 if successful, or PKI error code if not successful.

Remarks

NPKIGetServerDNSName can only be used after a successful call to NPKIGetServerIPAndDNSInfo (page 93), followed by a successful call to NPKIGetServerIPAddress (page 91).

See Also

NPKIGetServerDNSName (page 87), NPKIGetServerIPAddress (page 91)

NPKIGetServerInfo

Opens a connection to the specified server and sends a PKI ping NCP to determine supported values for the server (formerly NWPKIGetServerInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerInfo(
   const NPKIContext    context,
   const unicode       *serverDN,
   const nuint32        flags,
   pnuint32             keyGenerationalAlgorithms,
   pnuint32             signingAlgorithms,
   pnuint32             maxValidFromTime,
   pnuint32             maxValidToTime,
   pnuint32             caOperational,
   pnuint32             pathLength,
   pnuint32             reserved1,
   pnuint32             serverVersion,
   void                *reserved2);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

serverDN

(IN) Specifies the FDN of the server for which you want to get information. This must be a valid eDirectory server in the current tree.

flags

(IN) Specifies what information the ping requests. The following flags and are defined:

- PKI_CA_INFO—Retrieves information for creating or using a CA object.
- PKI_SERVER_INFO—Retrieves information for creating a server certificate.
- PKI_USER_INFO—Retrieves information for creating a user certificate.

keyGenerationAlgorithms

(OUT) Returns a bit mask that indicates which key generation algorithms are available on the server.

You can call NPKIGetAlgorithmInfo (page 73) for each of the algorithms to determine the maximum key size supported (this key generation algorithm is used as an argument in the NPKIGetAlgorithmInfo function to identify the maximum supported key sizes for key generation).

signingAlgorithms

(OUT) Returns a bit mask that indicates which signing algorithms are available on the server.

maxValidFromTime

(OUT) Returns the maximum starting validity period represented as the number of seconds since 00:00:00 UTC January 1, 1970. This time can be different depending on which flag is passed in the flag field. If the CA is installed and operational on the server specified in the call to NPKIGetServerInfo (page 88), this returns the time corresponding to the CA. See Remarks.

maxValidToTime

(OUT) Returns the maximum ending validity period represented as the number of seconds since 00:00:00 UTC January 1, 1970. This time can be different depending on which flag is passed in the flag field. If the CA is installed and operational on the server specified in the call to NPKIGetServerInfo (page 88), this returns the time corresponding to the CA. See Remarks.

caOperational

(OUT) Returns a bit mask that indicates whether a CA is installed and operational on the server specified in the call to NPKIGetServerInfo (page 88). The current possible bit values are as follows:

- PKI_NO_CA_PRESENT—The server does not host a CA.
- PKI_TREE_CA_PRESENT—The server hosts the organizational CA.

pathLength

(OUT) Returns the path length of the certificate authority certificates. For more information, see Section 4.1, "Basic Constraints Extension," on page 119.

NOTE: This parameter is valid only when the flags field is set to PKI_CA_INFO.

reserved1

Reserved for future use.

serverVersion

(OUT) Returns the version of the PKI.NLM, PKI.DLM or PKI.SO running on the server specified by the `serverDN` parameter.

reserved2

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or PKI error code if not successful.

PKI NCP Calls

0x2222 93 01 PKI Ping

Remarks

The *flags* parameter determines the set of information to acquire. The information returned from the ping is stored in context specific data values. You can call NPKIGetAlgorithmInfo (page 73) to get the supported key generation algorithm key sizes.

When creating server certificates, you must call "NPKIGetServerInfo" on page 88 on the server creating the key pair and the server hosting the CA. Use the greater of the `maxValidFromTime` and the lesser of the `maxValidToTime`.

If the key pair server and the CA server are the same, you only need to call NPKIGetServerInfo (page 88) once. For sample code, see GetServerInfo (../../../samplecode/ncslib_sample/GetServerInfo.cpp.html).

See Also

NPKIGetAlgorithmInfo (page 73)

NPKIGetServerIPAddress

Retrieves information about a specified IP address obtained by calling NPKIGetServerIPAndDNSInfo (page 93) (formerly NWPKIGetServerIPAddress).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerIPAddress(
   const NPKIContext             context,
   nuint32                       index,
   nuint16                      *ipLength,
   nuint8 const                **ipValue,
   unicode const               **ipNumber,
   nuint16                       *numberOfDNSNames);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Indicates which IP address is to be returned.

NOTE: `index` is 0 based.

`ipLength`

(OUT) Returns to the length of the data in `ipValue`.

`ipValue`

(OUT) Returns to the IP address in network byte order (hex).

`ipNumber`

(OUT) Returns to the IP address in Unicode format.

`numberOfDNSNames`

(OUT) Returns to the number of DNS names associated with the IP Address.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

Call NPKIGetServerIPAddress only after calling NPKIGetServerIPAndDNSInfo (page 93) successfully.

See Also

NPKIGetServerDNSName (page 87), NPKIGetServerIPAndDNSInfo (page 93)

NPKIGetServerIPAndDNSInfo

Discovers IP and DNS information about the specified server by querying DNS (formerlyNPKIGetServerIPAndDNSInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerIPAndDNSInfo(
   const NPKIContext              context,
   const unicode                 *serverDN,
   const pnuint32                 numberOfIPAddresses);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN of the eDirectory server.

`numberOfIPAddresses`

(OUT) Returns the number of IP addresses for the server.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

PKI NCP Calls

0x2222 93 14 GET IP AND DNS ADDRESSES

Remarks

Returns the number of IP addresses assigned to the server and other details about all of the IP addresses NPKIGetServerIPAddress (page 91) for each of the addresses.

See Also

NPKIGetServerDNSName (page 87), NPKIGetServerIPAddress (page 91)

NPKIGetServerUTCTime

Obtains the Universal Time Coordinated (UTC) time for a given server (formerly NWPKIGetServerUTCTime).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetServerUTCTime(
   const NPKIContext      context,
   const unicode         *serverDN,
   pnuint32               time);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN of the eDirectory server whose time you want. This must be a valid eDirectory server in the current tree.

`time`

(OUT) Points to the server's current time UTC time, represented as the number of seconds since 00:00:00 UTC January 1, 1970.

Return Values

Returns 0 if successful, or an eDirectory, PKI, or NetWare error code if not successful.

See Also

NPKIGetServerInfo (page 88)

NPKIGetTrustedRootInfo

Retrieves information about the specified trusted root (formerly NWPKIGetTrustedRootInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetTrustedRootInfo(
   const NPKIContext         context,
   nuint32                   index,
   unicode const           **name,
   unicode const           **validFrom,
   unicode const           **validTo,
   unicode const           **subjectName,
   nuit8 const             **certificate,
   pnuint32                  certificateLen);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

index

(IN) Specifies the Trusted Root object for which information is to be returned.

NOTE: *index* is 0 based.

name

(OUT) Returns the eDirectory FDN of the specified Trusted Root object.

validFrom

(OUT) Returns a Unicode string representation of the starting validity of the X.509 certificate stored in the specified Trusted Root object. Date is in the form YYYMMDDSS.

validTo

(OUT) Returns a Unicode string representation of the ending validity of the X.509 certificate stored in the specified Trusted Root object. Date is in the form YYYMMDDSS.

subjectName

(OUT) Returns a Unicode representation of the subject name of the X.509 certificate stored in the specified Trusted Root object.

certificate

(OUT) Returns the DER-encoded X.509 root (or CA) certificate stored in the specified Trusted Root object.

certificateLen

(OUT) Returns the size of the certificate.

Return Values

Returns 0 if successful, or an eDirectory error code if not successful.

Remarks

You must call NPKIFindTrustedRootsInContext (page 65) successfully before calling NPKIGetTrustedRootInfo. Trusted Root containers and Trusted Root objects provide a method to logically group, manage and access X.509 root (or certificate authority) certificates within a directory service.

See Also

NPKICreateTrustedRoot (page 39), NPKICreateTrustedRootContainer (page 41), NPKIFindTrustedRootsInContext (page 65)

NPKIGetWrappedServerKey

Obtains a server private key cryptographically wrapped in the server's key storage key (formerly NWPKIGetWrappedServerKey).

Syntax

```
#include "npki.h"

NWRCODE NPKIGetWrappedServerKey(
   const NPKIContext     context,
   const unicode        *serverDN,
   const unicode        *serverCertificateName,
   pnuint32              wrappedKeySize,
   nuint8 const        **wrappedKey);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN of the server for which you want to get a private key.

`serverCertificateName`

(IN) Specifies which server key to retrieve.

`wrappedKeySize`

(OUT) Specifies the size of the cryptographically wrapped private key.

`wrappedKey`

(OUT) Points to the cryptographically wrapped private key.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

See Also

NPKICreateServerCertificate (page 34), NPKIFindServerCertificateNames (page 62), NPKIServerCertificateName (page 104)

NPKIImportCAKey

Imports an organization's CA private key and corresponding certificates from a PFX format (also known as PKCS #12) to a CA object (formerly NWPKIImportCAKey).

Syntax

```
#include "npki.h"

void NPKIImportCAKey(
   const NPKIContext           context,
   const unicode              *hostServerDN,
   const unicode              *organizationlCAName,
   const unicode              *password,
   const nuint32               flags,
   const nuint32               pfxSize,
   const nuint8               *pfx);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`hostServerDN`

(IN) Specifies the FDN of the server to host the CA.

`organizationlCAName`

(IN) Specifies the name of the organizational CA object certificate and private key you want to import. If the CA object does not exist, one is created.

`password`

(IN) Specifies the password used to decrypt the private key and certificates.

`flags`

(IN) Specifies options for importing the server key and certificate. The flags currently defined are:PKI_OVERWRITE—Overwrites any information currently associated with this certificate name for the indicated server.

`pfxSize`

(IN) The size of the data in `pfx`.

`pfx`

(IN) Specifies the PKCS #12 encoded data to import.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or a PKI error code if not successful.

PKI NCP Calls

0x2222 93 10 Write Key

See Also

"NPKIExportCAKey" on page 54

NPKIImportServerKey

Imports a server's private key and corresponding certificates from a PFX format (also known as PKCS #12) to a KMO (formerly NWPKIImportServerKey).

Syntax

```
#include "npki.h"

void NPKIImportServerKey(
   const NPKIContext           context,
   const unicode              *serverDN,
   const unicode              *certificateName,
   const unicode              *password,
   const nuint32               flags,
   const nuint32               pfxSize,
   const nuint8               *pfx);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN eDirectory server. This must be a valid eDirectory server in the current tree.

`certificateName`

(IN) Specifies the name of the certificate and private key you want to import. If the KMO corresponding to *certificateName* does not exist, one is created.

`password`

(IN) Specifies the password used to decrypt the private key and certificates.

`flags`

(IN) Specifies options for importing the server key and certificate. The flags currently defined are:PKI_OVERWRITE—Overwrites any information currently associated with this certificate name for the indicated server.

`pfxSize`

(IN) The size of the data pointed in `pfx`.

`pfx`

(IN) Specifies to the PKCS #12 encoded data to import.

Return Values

Returns 0 if successful, or an eDirectory, NICI, or a PKI error code if not successful.

PKI NCP Calls

0x2222 93 10 Write Key

See Also

NPKIExportServerKey (page 56)

NPKINickName

Obtains a pointer to a certificate nickname (formerly NWPKINickName).

Syntax

```
#include "npki.h"

NWRCODE NPKINickName(
   const NPKIContext    context
   nuint32              index,
   unicode const      **nickName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Specifies which nickname is to be returned.

NOTE: *index* is 0 based.

`nickName`

(OUT) Returns the specified nickname.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

You must call NPKIReadAllNickNames (page 103) successfully before calling NPKINickName.

See Also

NPKIFindUserCertificates (page 66), NPKIReadAllNickNames (page 103)

NPKIReadAllNickNames

Reads the certificate nicknames for *userDN* and stores them in context-specific data values (formerly NWPKIReadAllNickNames).

Syntax

```
#include "npki.h"

NWRCODE NPKIReadAllNickNames(
   const NPKIContext    context,
   const unicode       *userDN,
   pnuint32             numberOfNickNames);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

userDN

(IN) Specifies the FDN of the user for which you want to read certificate names. *userDN* must be a valid User object in the current tree.

numberOfNickNames

(OUT) Returns the number of nicknames of certificates for the specified user.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

Calls to NPKINickName (page 102) can be made to retrieve the nicknames. For sample code, see ReadUserNicknames (../../../samplecode/ncslib_sample/ReadUserNicknames.cpp.html).

See Also

NPKIFindUserCertificates (page 66), NPKINickName (page 102)

NPKIServerCertificateName

Obtains a server certificate name (formerly NWPKIServerCertificateName).

Syntax

```
#include "npki.h"

NWRCODE NPKIServerCertificateName(
   const NPKIContext    context,
   nuint32              index,
   unicode const      **serverCertificateName);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

index

(IN) Specifies which server certificate name is to be returned.

NOTE: *index* is 0 based.

serverCertificateName

(OUT) Points to the specified server certificate name.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

You must call NPKIFindServerCertificateNames (page 62) successfully immediately before calling NPKIServerCertificateName.

See Also

NPKIFindServerCertificateNames (page 62)

NPKIServerNames

Obtains the specified eDirectory server's leaf name and FDN (formerly NWPKIServerNames).

Syntax

```
#include "npki.h"

NWRCODE NPKIServerNames(
   const NPKIContext    context,
   nuint32              index,
   unicode const      **serverDN,
   unicode const       **serverName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Indicates which server name is to be returned.

NOTE: *index* is 0 based.

`serverDN`

(OUT) Returns the eDirectory FDN of the server.

`serverName`

(OUT) Returns the leaf name of the server.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

You must call either NPKIFindKeyGenServersForUser (page 60) or NPKIFindServersInContext (page 63) before calling NPKIServerNames. You can call this function repeatedly to get all of the server names.

See Also

NPKIFindKeyGenServersForUser (page 60), NPKIFindServersInContext (page 63)

NPKISetTreeName

Sets the given tree name in to the context (formerly NWPKISetTreeName).

Syntax

```
#include "npki.h"

NWRCODE NPKISetTreeName(
   const NPKIContext    context,
   const unicode       *treeName);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`treeName`

(IN) Specifies the tree name. This must be a valid eDirectory tree.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

NPKIStoreServerCertificates

Stores server certificates after a successful call to NPKICreateServerCertificate (page 34) (formerly NWPKIStoreServerCertificates). (Being deprecated.)

Syntax

```
#include "npki.h"

NWRCODE NPKIStoreServerCertificates(
   const NPKIContext     context,
   const unicode        *serverDN,
   const unicode        *certificateName,
   const nuint32         flags,
   const nuint32         trustedRoot,
   const pnuint8         certificate,
   const nuint32         certificateLen,
   void                 *reserved1,
   void                 *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Specifies the FDN of the eDirectory server (that is, the server for which the certificate(s) are stored).

`certificateName`

(IN) Specifies which server certificate you want to store.

`flags`

(IN) Specifies which certificates are stored. The flags currently defined are:

- PKI_CHAIN_CERTIFICATE—Store the certificate chain.
- PKI_TRUSTED_ROOT_CERTIFICATE—Store the trusted root.
- PKI_SELF_SIGNED_CERTIFICATE—Store the self-signed certificate.
- PKI_OBJECT_KEY_CERTIFICATE—Store the object certificate.
- PKI_WAIVE_SUBJECT_NAME_IN_CERTIFICATE—Normally NPKIStoreServerCertificates (page 107) checks that the requested name and the subject name in the certificate match. This optional flag waives the check, enabling the certificate to be stored even if the requested name and certificate name are dissimilar.

NOTE: The flags PKI_CHAIN_CERTIFICATE, PKI_TRUSTED_ROOT_CERTIFICATE, and PKI_SELF_SIGNED_CERTIFICATE are mutually exclusive. In addition, PKI_OBJECT_KEY_CERTIFICATE and PKI_TRUSTED_ROOT_CERTIFICATE are also mutually exclusive.

`trustedRoot`

(IN) Specifies which certificate to mark as the trusted root. Use one of the following defines:

- PKI_ORG_CA_CERTIFICATE—Use the self-signed organizational certificate as the trusted root. This is the most commonly used option.

  NOTE: This is the default flag developers typically should use.

- PKI_NOVELL_CERTIFICATE—Use the Novell Root Certifier Certificate as the trusted root. (Use this option only if your software can natively understand and process the Novell Security Attributes extension.)

  NOTE: If PKI_NOVELL_CERTIFICATE is used, the developer's relying software must be configured to handle the Novell Security Attributes extension (see Section 4.16, “X.509 Extensions,” on page 130).

`certificate`

(IN) (Optional) Specifies a DER-encoded X.509 certificate.

NOTE: If the `certificate` parameter is not used, you must call NPKIGetCACertificates (page 75) immediately before calling NPKIStoreServerCertificates.

`certificateLen`

(IN) (Optional) Specifies the length of the certificate, if present.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory, PKI, or NetWare error code if not successful.

PKI NCP Calls

0x2222 93 07 Store Certificate

Remarks

IMPORTANT: NPKIStoreServerCertificates is being deprecated because it can only handle a chain of two certificates. Use NPKICertificateList (page 22) and NPKIStoreServerCertificatesFromCertificateList (page 110) to replace NWPKIStoreServerCertificates.

Two of the three modes of calling NPKICreateServerCertificate (page 34) require subsequent calls to NPKIStoreServerCertificates (page 107).

In the two server mode, after successfully calling NPKICreateServerCertificate, you should call NPKIGetCACertificates (page 75) to retrieve the CA's self-signed certificate. Then you should call NPKIStoreServerCertificates to store the certificates.

NPKIStoreServerCertificates combines the CA's object certificate and certificate chain to form the certificate chain for the server.

In the external certificate authority mode, two calls to NPKIStoreServerCertificates should be made. One call should store the certificate chain and the other should store the newly created certificate. The `certificates` and `certificateLen` parameters provide the capability to send in a certificate to be stored.

See Also

NPKICertificateList (page 22), NPKICreateServerCertificate (page 34), NPKIFindServerCertificateNames (page 62), NPKIGetCACertificates (page 75), NPKIServerCertificateName (page 104), NPKIStoreServerCertificatesFromCertificateList (page 110)

NPKIStoreServerCertificatesFromCertificateList

Stores server certificates from an internal certificate chain structure.

Syntax

```
#include "npki.h"

NWRCODE NPKIStoreServerCertificatesFromCertificateList(
   const NPKIContext    context,
   const unicode       *serverDN,
   const unicode       *certificateName,
   const nuint32        flags,
   const nuint32        trustedRootIndex,
   void                *reserved1,
   void                 *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`serverDN`

(IN) Points to the distinguished name of the server.

`certificateName`

(IN) Points to the name of the certificate.

`flags`

(IN) Reserved Pass in zero.

`trustedRootIndex`

(IN) Specifies the index number of the trusted root.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

PKI NCP Calls

0x2222 93 07 Store Certificate

Remarks

Two of the three modes of calling NPKICreateServerCertificate (page 34) require subsequent calls to NPKICertificateList (page 22) and NPKIStoreServerCertificatesFromCertificateList (page 110). In the two server mode, after successfully calling NPKICreateServerCertificate (page 34), you should call NPKIGetCACertificates (page 75) successfully to retrieve the CA's self-signed certificate. Call NPKICertificateList (page 22) to add the self-signed certificate to the list. Then call NPKICertInfo (page 24) to retrieve the newly created server certificate. Next call NPKICertificateList to add it to the list, then call NPKIStoreServerCertificatesFromCertificateList to store the certificates.

NPKIStoreServerCertificatesFromCertificateList (page 110) combines the CA's object certificate and certificate chain to form the certificate chain for the server.In the external certificate authority mode, calls to NPKICertificateList should be made for each of the certificates to store the whole certificate chain from root to leaf. Then call NPKIStoreServerCertificatesFromCertificateList to store the newly formed chain to the KMO.

See Also

NPKICertificateList (page 22), NPKICertInfo (page 24), NPKICreateServerCertificate (page 34), NPKIGetCACertificates (page 75)

NPKIStoreUserCertificate

Stores a certificate on a user object (formerly NWPKIStoreUserCertificate).

Syntax

```
#include "npki.h"

NWRCODE NPKIStoreUserCertificate(
   const NPKIContext    context,
   const unicode       *userDN,
   const unicode       *nickName,
   const unicode       *signerDN,
   const nuint32        flags,
   const pnuint8        cert,
   const nuint32        certSize,
   const nuint32        vendorID,
   const pnuint8        privateKey,
   const nuint32        privateKeySize,
   void                *reserved1,
   void                *reserved2);
```

Parameters

context

(IN) Specifies the NPKI context for the request.

userDN

(IN) Specifies the FDN of a User object. This must be a valid eDirectory user object in the current tree.

nickName

(IN) Specifies the certificate nickname. This name is used to identify the key pair and associated certificate. This name must be unique for the specified user.

signerDN

(IN) Specifies the FDN of the eDirectory object that signed the certificate.

If the certificate is an external certificate, `signerDN` can be set to point to the trusted root object that contains the certificate of the signing CA, or it can be set to the user object.

flags

(IN) Specifies options when storing user certificates. If the key pair was generated by the Novell Certificate Server and the private key is stored in eDirectory, the flag PKI_INTERNAL_KEY_PAIR should be used. If the key pair was generated external to the Novell Certificate Server, the flag PKI_EXTERNAL_KEY_PAIR should be used.

cert

(IN) Specifies the DER-encoded X.509 certificate that you want to store. This parameter can be a NULL if you called NPKICreateUserCertificate (page 42) immediately before this function and the error PKI_E_ADD_CERTIFICATE was returned. The flag PKI_INTERNAL_KEY_PAIR must be set when `cert` is NULL.

`certSize`

(IN) Specifies the size of the certificate.

This parameter can be 0 if you called NPKICreateUserCertificate (page 42) immediately before this function and the error PKI_E_ADD_CERTIFICATE was returned. The flag PKI_INTERNAL_KEY_PAIR must be set when `certSize` is 0.

`vendorID`

(IN) Specifies which vendor supplied the certificate. If the flag PKI_INTERNAL_KEY_PAIR is set, this parameter is ignored, and the `vendorID` is set to PKI_VENDOR_NOVELL.

`privateKey`

(IN) Not implemented in this release. Set to NULL.

`privateKeySize`

(IN) Not implemented in this release. Set to 0.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

See Also

NPKICreateUserCertificate (page 42), NPKIFindUserCertificates (page 66)

NPKIUserCertInfo

Obtains information about a user certificate (formerly NWPKIUserCertInfo).

Syntax

```
#include "npki.h"

NWRCODE NPKIUserCertInfo(
   const NPKIContext    context,
   const nuint32        index,
   unicode const      **nickName,
   pnuint32             certSize,
   nuint8 const       **cert,
   pnuint32             certStatus,
   pnuint32             certChainSize,
   nuint8 const       **certChain,
   pnuint32             vendorID,
   void                *reserved1,
   void                 *reserved2);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`index`

(IN) Specifies which certificate is to be returned.

NOTE: *index* is 0 based.

`nickName`

(OUT) Returns the certificate nickname. This name is used to identify the key pair and associated certificate. This name must be unique for the specified user.

`certSize`

(OUT) Returns size of the specified certificate.

`cert`

(OUT) Returns the specified DER-encoded X.509 certificate.

`certStatus`

(OUT) Returns status of the certificate.

`certChainSize`

(OUT) Not implemented in this release.

`certChain`

(OUT) Not implemented in this release.

`vendorID`

(OUT) Returns vendor ID associated with the certificate.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

You must call either NPKICreateUserCertificate (page 42) or NPKIFindUserCertificates (page 66) successfully immediately before calling this function.

See Also

NPKICreateUserCertificate (page 42), NPKIFindUserCertificates (page 66)

NPKIVerifyCertificateWithTrustedRoots

Constructs a certificate chain starting with the specified certificate and using all of the Trusted Root objects within the specified Trusted Root container (formerly NWPKIVerifyCertificateWithTrustedRoots).

Syntax

```
#include "npki.h"

NWRCODE NPKIVerifyCertificateWithTrustedRoots(
   const NPKIContext              context,
   const pnuint8                  certificate,
   const nuint32                  certificateLen,
   const unicode                 *TRContextDN,
   void                          *reserved1,
   nuint32                        flags,
   pnuint32                       cRLReason,
   pnuint32                       cRLHoldInstruction,
   time_t                        *cRLRevocationTime,
   time_t                        *cRLInvalidityDateTime,
   pnuint32                       certInvalidityReason,
   void                          *reserved4);
```

Parameters

`context`

(IN) Specifies the NPKI context for the request.

`certificate`

(IN) Specifies the DER-encoded X.509 certificate you want to verify.

`certificateLen`

(IN) Specifies the size of the certificate.

`TRContextDN`

(IN) Specifies the FDN of the Trusted Root container that is to be searched.

`reserved1`

Reserved for future use

`flags`

(IN) Specifies whether to verify the certificate, check certificate revocation, both, or neither. For related flag definitions, see Section 4.12, “NPKIx509 Certificate Invalidity Reasons,” on page 127 and Section 4.13, “NPKIx509 CRL Hold Types,” on page 128.

`cRLReason`

(OUT) Returns the reason code, if the certificate has been revoked (that is, the reason the certificate has been revoked; private key compromised, affiliate change, superseded, etc.). This field is set only if the return code is set to PKI_E_CERT_INVALID and the `certInvalidityReason` is set to NPKIx509Invalid_Certificate_On_CRL.

`cRLHoldInstruction`

(OUT) Returns the hold instruction from the CRL, if the certificate has been revoked and the reason code is `certificateHold`, points to the hold instruction from the CRL. This field will be set only if the return code is set to PKI_E_CERT_INVALID and the `certInvalidityReason` is set to NPKIx509Invalid_Certificate_On_CRL and the `cRLReason` is set to PKI_CERTIFICATE_HOLD.

`cRLRevocationTime`

(OUT) Returns the date the certificate became invalid. This field is set only if the return code is set to PKI_E_CERT_INVALID and the `certInvalidityReason` is set to NPKIx509Invalid_Certificate_On_CRL.

`cRLInvalidityDateTime`

(OUT) Returns the date the CRL becomes invalid.

`certInvalidityReason`

(OUT) Returns the reason why the certificate is invalid. (that is, revoked, invalid issuer, unreadable extensions, expired, etc.). This field is set only if the return code is set to PKI_E_CERT_INVALID.

`reserved4`

Reserved for future use.

Return Values

Returns 0 if successful, or an eDirectory or PKI error code if not successful.

Remarks

The chain is considered complete once a self-signed certificate has been found. Once the complete certificate chain has been constructed, it is to verified. Certificate revocation checking is supported.

For sample code, see VerifyWithTrustedRoot (../../../samplecode/ncslib_sample/VerifyWithTrustedRoot.cpp.html).

See Also

NPKICreateTrustedRootContainer (page 41), NPKICreateTrustedRoot (page 39), NPKIFindTrustedRootsInContext (page 65), NPKIGetTrustedRootInfo (page 95)

NPKIVersionInfo

Obtains the version info of the client module, NPKIAPI.

Syntax

```
#include "npki.h"

NWRCODE NPKIVersionInfo(
   nuint32       *versionNumber,
   void          *reserved1,
   void          *reserved2);
```

Parameters

`versionNumber`

(OUT) The version number of the client module. For Windows NT/2000 the module is npkiapi.dll. For Netware it is npkiapi.nlm, and for Linux, Solaris, and AIX it is libnpkiapi.so.

`reserved1`

Reserved for future use.

`reserved2`

Reserved for future use.

Return Values

Returns 0 if successful, or a PKI error code if not successful.

Remarks

A PKIContext does not need to be created to call this function. For sample code, see VersionInfo (../../../samplecode/ncslib_sample/VersionInfo.cpp.html).

4 Defines

Novell® Certificate Server™ Library for C Version 2 interfaces, prototypes, and data types are defined in the Novell header files, npki.h, and npki_ver.h. The data types and structures used for verification are found in nverify.h. Their use and definition are found in npkit.doc.

This section contains the following topics:

4.1 Basic Constraints Extension

The X.509 basic constraints extension is used to specify whether the certificate is for a certificate authority (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 to infinite. If the `value` is zero, it means that the CA cannot create other CAs but it can still create end entity objects (that is, user and server certificates). If the `value` is one, it means that at most a CA one level below this CA can be created, etc. If the `pathLenConstraint` is not specified it means the value is infinite and there is no restrictions on the number of levels of CAs that can be created.

CAs must have the basic constraints extension encoded. Certificates for non-CAs should not have the basic constraints extension encoded.

The basic constraints extension uses the general purpose extension structure Section 5.2, "NPKI_Extension," on page 134 described in the Section 5.2.1, "General Purpose Extension Structure," on page 134.

`value` might or might not be present. If `value` is present, it should be one nuint32 encoded as `pathLenConstraint` in the extension. If no path length constraint is desired (that is, a value of infinite), `length` should be set to 0 and `value` should not be present.

There is one extension specific flag defined for the basic constraints extension:

Value	Name	Description
0x0100	X509_BASIC_CONSTRAINTS_CA	Specifies that the certificate is for a CA.

4.2 Date Flags

The date flags determine the ASN.1 encoding of the validity period in the certificate.

Value	Name	Description
0x00	DEFAULT_YEAR_ENCODING	There is nothing specific to do.
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 X.509 default for date encoding (see RFC 2459 for more details).

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.

Value	Name	Description
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.
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

The X.509 key usage extension is used to specify how a key is used. When an application goes through the verification process, it normally checks that the key is only being used for an intended purpose.

The key usage extension uses the general purpose extension structure Section 5.2.1, "General Purpose Extension Structure," on page 134.

No additional flags are defined for this extension.

`value` must be one `nuint16` where each bit is a key usage. Any combination of key usages may be used, but not all are appropriate combinations or are appropriate for all types of keys.

The following key usages are defined:

Value	Name	Description
0x8000	X509_KEY_USAGE_DIGITAL_SIGNATURE	Designate that the key is used to create digital signatures.
0x4000	X509_KEY_USAGE_NON_REPUDIATION	Designate that the key will be used for non-repudiation. This type of key usually has legal ramifications.
0x2000	X509_KEY_USAGE_KEY_ENCIPHERMENT	Designates that the key will be used to encrypt other keys.
0x1000	X509_KEY_USAGE_DATA_ENCIPHERMENT	Designates that the key will be used to directly encrypt data.
0x0800	X509_KEY_USAGE_KEY_AGREEMENT	Not valid for RSA keys.
0x0400	X509_KEY_USAGE_KEY_CERT_SIGN	Designates that the key will be used to sign certificates.

Value	Name	Description
0x0200	X509_KEY_USAGE_CRL_SIGN	Designates that the key will be used to sign CRLs.
0x0100	X509_KEY_USAGE_ENCIPHER_ONLY	Not valid for RSA keys.
0x0080	X509_KEY_USAGE_DECIPHER_ONLY	Not valid for RSA keys.

4.5 Novell Security Attributes Extension

The Novell Security Attributes extension is used to specify the cryptographic qualities of the key and the environment in which the key was generated. In addition, it can be used to identify the enterprise that the subject (of the X.509 certificate) belongs to.

The Novell Security Attributes extension specific flags are optional flags that can be used to specify values to be encoded into the Novell Security Attributes extension. If no extension specific flags are set, the lowest cryptographic qualities are encoded.

The Novell Security Attributes extension uses the general purpose extension structure as described in Section 5.2.1, "General Purpose Extension Structure," on page 134.

For this release, *value* must not be present, and *length* should be set to 0.

This section contains the following topics:

- Section 4.5.1, "Mutually Exclusive Flags," on page 122
- Section 4.5.2, "Additional Flags," on page 122

4.5.1 Mutually Exclusive Flags

The mutually exclusive flags used in the Novell Security Attributes extension are defined below:

Value	Name	Description
0x00100	NOVELL_EXTENSION_SERVER_DEFAULT	Specifies that the key pair is for a server.
0x00200	NOVELL_EXTENSION_USER_DEFAULT	Specifies that the key pair is for a user.
0x00400	NOVELL_EXTENSION_ORGCA_DEFAULT	Specifies that the key pair is for the Organizational CA.

4.5.2 Additional Flags

An additional flag used in the Novell Security Attributes extension is defined below:

Value	Name	Description
0x10000	NOVELL_EXTENSION_EXTRACTABLE_KEY	Specifies that the private key can be extracted from the Novell International Cryptographic Infrastructure (NICI). Setting this flag reduces the cryptographic quality. This flag only applies to keys generated by Novell PKIS.
0x00200	NOVELL_EXTENSION_USER_DEFAULT	Specifies that the key pair is for a user.
0x00400	NOVELL_EXTENSION_ORGCA_DEFAULT	Specifies that the key pair is for the Organizational CA.

4.6 NPKI Context Definitions

Value	Name	Description
nuit32	NPKIContext	Definition of the NPKIContext.
***	NPKI_INVALID_CONTEXT	Invalid context number.

4.7 NPKIGetServerCertificateStatus Defines

One of the following defined values is returned in the flags parameter after a successful call to NPKIGetServerCertificateStatus (page 85).

Value	Name	Description
0	KMO_EMPTY	No certificates or key pair have been stored.
1	KMO_KEY_PAIR_PRESENT	A key pair has been stored, but no certificates have been stored.
2	KMO_TRUSTED_ROOT_PRESENT	A key pair and the root certificate have been stored, but the object certificate has not been stored.
3	KMO_CERTIFICATE_PRESENT	All certificates as well as the key pair have been stored, and the object is in working order.
0xffffffff	KMO_INVALID_STATUS	The object is in an invalid state.

4.8 NPKIExportCAKey Flags

The following flags are used to specify which set of information to return by a successful call to NPKIExportCAKey (page 54).

Value	Name	Description
0x04 0x10 0x01	PKI_CA_KEY_AND_CERTS	Use this flag when exporting the CA self-signed certificate, public certificate, and the CA's chain.

4.9 NPKIExportServerKey Flags

These flags determine which certificates are obtained by calls to: NPKIGetCACertificates (page 75), NPKIGetServerCertificates (page 83), “NPKIExportServerKey” on page 56, “NPKIExportUserKey” on page 58 and NPKIStoreServerCertificates (page 107). These flags typically are used for more advanced applications.

Value	Name	Description
0x01	PKI_OBJECT_KEY_CERTIFICATE	Use this flag when exporting or retrieving information about the object certificate.
0x02	PKI_TRUSTED_ROOT_CERTIFICATE	Use this flag when exporting or retrieving information about the trusted root certificate. **IMPORTANT:** This flag is not for use with `NPKIExportServerCert`. Use the flag the flag PKI_SELF_SIGNED_CERTIFICATE instead.
0x04	PKI_CHAIN_CERTIFICATE	Use this flag when exporting or retrieving information about the certificate chain.
0x10	PKI_SELF_SIGNED_CERTIFICATE	This flag is not for use with NPKIGetServerCertificates (page 83). Use this flag when exporting or retrieving information on the self-signed certificate.

4.10 NPKIGetServerInfo Defines and Flags

- Section 4.10.1, “Server Information Flags,” on page 124
- Section 4.10.2, “Key Generation Algorithms Defines,” on page 125
- Section 4.10.3, “Signing Algorithms Defines,” on page 125
- Section 4.10.4, “Key Pair Storage Defines,” on page 125
- Section 4.10.5, “CA Operational Defines,” on page 126

4.10.1 Server Information Flags

The following flags are used to specify which set of information to obtain by calling NPKIGetServerInfo (page 88). For sample code, see GetServerInfo (../../../samplecode/ncslib_sample/GetServerInfo.cpp.html).

Value	Name	Description
0x01	PKI_CA_INFO	Use when querying for information about creating or using a certificate authority.
0x02	PKI_USER_INFO	Use when querying for information about creating a user certificate.
0x03	PKI_SERVER_INFO	Use querying for information about creating a server certificate.

4.10.2 Key Generation Algorithms Defines

One of the following values will be returned in the keyGenerationAlgorithms parameter by a successful call to NPKIGetServerInfo (page 88). (Currently, only the RSA algorithm is supported.)

Value	Name	Description
0x01	PKI_RSA_ALGORITHM	The RSA algorithm is supported.

4.10.3 Signing Algorithms Defines

One of the following values will be returned in the *signingAlgorithms* parameter by a successful call to NPKIGetServerInfo (page 88). (Currently only RSA algorithms are supported.)

Value	Name	Description
0x00	PKI_UNKNOWN_ALGORITHM	The algorithm is unknown and not supported.
0x01	PKI_SIGN_WITH_RSA_AND_MD2	The MD2 with RSA encryption signing algorithm is supported.
0x02	PKI_SIGN_WITH_RSA_AND_MD5	The MD5 with RSA encryption signing algorithm is supported.
0x04	PKI_SIGN_WITH_RSA_AND_SHA1	The SHA1 with RSA encryption signing algorithm is supported.

4.10.4 Key Pair Storage Defines

The following flags are used to specify which set of information to return from a successful call to NPKIStoreUserCertificate (page 112).

Value	Name	Description
0x00000001	PKI_INTERNAL_KEY_PAIR	The key pair was generated by Novell PKI service and the private key is stored in eDirectory™.
0x00000002	PKI_EXTERNAL_KEY_PAIR	The key pair was generated external to the Novell PKI service.
0x00004000	PKI_DONT_CHECK_NICKNAME_UNIQUENESS	It is allowable to have more than one user certificate belonging to a user with the same nickname.

4.10.5 CA Operational Defines

One of the following values is returned in the *caOperational* parameter by a successful call to NPKIGetServerInfo (page 88).

Value	Name	Description
0x00	PKI_NO_CA	A CA is either not installed and/or not operational on the specified server.
0x01	PKI_ORGANIZATIONAL_CA	An organizational CA is installed and operational on the specified server.
0x02	PKI_SUB_ORGANIZATIONAL_CA	An organizational CA is installed and operational on the specified server (currently not supported).

4.11 NPKI_Version Values

The NPKI NDK Version number can be used to compare with the return value of NPKIVersionInfo (page 118) to determine if the version of the NPKI 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.

Value	Constant	Description
0x00020706	PKIS_VERSION_TWO_SEVEN_SIX	PKIS NDK version 2.7.6.
PKIS_VERSION_TWO_SEVEN_SIX	NPKI_VERSION	The current NPKI version 2.7.6.

4.12 NPKIx509 Certificate Invalidity Reasons

The following section describes the certificate invalidity reason flags:

4.12.1 flags

The following flags are used to specify why a certificate may be invalid. For use with the cRLReason field in the function NPKIVerifyCertificateWithTrustedRoots (page 116).

Value	Name	Description
0x0000000	NPKIx509CertificateValid	The certificate is valid.
0x0000001	NPKIx509Invalid_System_Error	The system is unstable and should be rebooted.
0x0000002	NPKIx509Invalid_Decode_Error	There was an ASN1 decoding problem.
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 start date is in the future.
0x0000005	NPKIx509Invalid_Expired	The end date is in the past.
0x0000006	NPKIx509Invalid_Issuer_Not_CA	The issuer is not a valid CA.
0x0000007	NPKIx509Invalid_Path_Length	The X.509 basic constraints extension path length has been violated.
0x0000008	NPKIx509Invalid_Unknown_Critical_Extension	There was a critical extension that could 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 certificate revocation list (CRL).
0x000000B	NPKIx509Invalid_Certificate_On_CRL	One of the certificates in the chain is on a CRL.
0x000000C	NPKIx509Invalid_Cant_Process_CDP	The certificate contained 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 was not valid for this certificate.
0x000000F	NPKIx509Invalid_Expired_CRL.	The CRL has expired.

Value	Name	Description
0x0000010	NPKIx509Invalid_CRL_Issuer_Name	The issuer name of the CRL identified in the certificate does not match the issuer name in the actual CRL retrieved.
0x0000011	NPKIx509Invalid_Issuer_Not_Trusted	Indicates that one or more of the certificates in the certificate chain does not exist in the specified trusted root container. This error code can only be returned by a call to NPKIVerifyCertificateWithTrustedRoots (page 116).
0x0000012	NPKIx509Invalid_CDP_Exists_Did_Not_Check_CRL	This is an advisory flag. The Certificate Distribution Point (CDP) exists but the CRL was not checked because the caller of the function requested that it not be checked.
0x0000013	NPKIx509Invalid_Invalid_Signature	The signature on the CRL is invalid.

4.13 NPKIx509 CRL Hold Types

The following section describes the CRL hold flags:

4.13.1 Certificate Hold Flags

The following flags are used to specify what to do if a certificate is on hold. For use with the cRLHoldInstruction field in the functions NPKIVerifyCertificateWithTrustedRoots (page 116).

Value	Name	Description
0	PKI_HOLD_INSTRUCTION_NONE	There is nothing specific to do.
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 though it were revoked.

4.14 Private Key Flags

The following tables provide the general and private key flags:

- Section 4.14.1, "General Private Key Flag," on page 128
- Section 4.14.2, "Optional Private Key Flag," on page 129

4.14.1 General Private Key Flag

Use the following private key flag for creating certificates.

Value	Name	Description
0x0002	PRIVATE_KEY	Use for all certificates.

4.14.2 Optional Private Key Flag

Value	Name	Description
0x0004	PRIVATE_KEY _EXTRACTABLE	Use to allow a key to be extracted out of NICI. This is valid for all certificates.

NOTE: When using the PRIVATE_KEY_EXTRACTABLE flag and including the Novell Security Attributes™ Extension, it's necessary to bitwise-OR the extractable option (that is, NOVELL_EXTENSION_EXTRACTABLE_KEY) along with the appropriate Novell attribute to the flags field in the Novell Security Attributes extension.

4.15 Public Key Flags

The following tables provide the public key flags:

- Section 4.15.1, "Certificate Authority Public Key Flags," on page 129
- Section 4.15.2, "End Entity Certificate Creation Public Key Flags," on page 129
- Section 4.15.3, "Optional Certificate Creation Public Key Flags," on page 130
- Section 4.15.4, "Server Private Key and Certificate Flag," on page 130

4.15.1 Certificate Authority Public Key Flags

Use the following flag for creating an Organizational Certificate Authority (CA):

Value	Name	Description
0x0001 | 0x0002 | 0x0020	PUBLIC_KEY_ORGANIZATIONAL_CA	Use when creating a CA.

4.15.2 End Entity Certificate Creation Public Key Flags

Use one of the following flags for creating user and server certificates:

Value	Name	Description
0x0002 | 0x0020 | 0x0100	PUBLIC_KEY_SINGLE_SERVER	Use when the key generation server is the same as the CA server.
0x0004 | 0x0010 | 0x0100	PUBLIC_KEY_TWO_SERVER	Use when the key generation server is not the same as the CA server.
0x0004 | 0x0010 | 0x0100	PUBLIC_KEY_EXTERNAL_CA	Use when the CA is external to Novell.

4.15.3 Optional Certificate Creation Public Key Flags

Value	Name	Description
0x00001000	PKI_CUSTOM_SUBJECT_NAME	Use when the subject name is not the default.
0x100	PKI_WAIVE_SUBJECT_NAME_IN_CERTIFICATE	Use when the subject name of the PKCS #10 CSR is not the subject name returned in the X.509 certificate.

4.15.4 Server Private Key and Certificate Flag

Value	Name	Description
0x01	PKI_OVERWRITE	Use to allow import of a server private key and certificate.

4.16 X.509 Extensions

The extensions of a X.509 certificate provide a generic way to include information in the certificate. Currently the API provides explicit support for four X.509 extensions: Key Usage, basic constraints, subject alternative name, and the Novell Security Attributes. In addition, the API currently supports the ability to include any generic ASN.1 encoded extensions when generating server and user certificates.

NOTE: Creating an ASN.1 encoded extension is an advanced operation, requiring detailed knowledge of ASN.1 and X.509 extensions. However, existing ASN.1 encoded extensions may be used without such detailed knowledge. To review an example that uses the parameters to include in an extended key usage extension on the user certificate as it is created, see UserExtendedKeyUsage (../../../samplecode/ncslib_sample/UserExtendedKeyUsage.cpp.html).

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. Also see Section 5.2.1, "General Purpose Extension Structure," on page 134. The following table describes the general purpose extension flags:

4.16.1 General Purpose Extension Flags

The following are a list of general purpose extension flags:

Value	Name	Description
0x0000	PKI_EXTENSION_INCLUDE	The extension is included in the certificate.
0x0001	PKI_EXTENSION_DONT_INCLUDE	Excludes the extension from the certificate.

Value	Name	Description
0x0002	PKI_EXTENSION_CRITICAL	Use to set the extension as critical in the certificate. **NOTE:** If an extension is set to critical, application software should understand the extension, or fail verification of the certificate.)

4.17 Subject Alternative Name Types (obsolete, 3/2005)

The subject alternative name type determined which encoding format was used to encode the alternative name and is now deprecated and replaced by Section 4.3, "General Name Type Extensions," on page 120. As specified by the X.509 standard, three of the name types were encoded as IA5String, which is the same as ASCII (`rfc822Name`, `dNSName`, and `uniformResourceIdentifier`).

When using any of the three forms specified above, the `value` must contain the Unicode representation of the IA5String and the `length` field must contain the number of bytes in the Unicode string including the NULL terminator.

NOTE: The names are specified in Unicode instead of IA5String because all other parameters in the API are Unicode. However, only IA5String characters are supported.

When using any other than the three forms specified above, `value` field must contain the data structures defined in the X.509 document RFC 2459. (This means that other forms required you to do any ASN.1 encoding.)

Also see Section 5.3, "Subject Alternative Names Extension," on page 134.

The following subject alternative name types were defined:

Value	Name	Description
0x0000	X509_SUBJECT_ALT_NAME_OTHER_NAME	The alternative name must be encoded as an `OtherName` sequence as specified in RFC 2459.
0x0001	X509_SUBJECT_ALT_NAME_RFC822_NAME	The alternative name must be a Unicode representation of an IA5String.
0x0002	X509_SUBJECT_ALT_NAME_DNS_NAME	The alternative name must be a Unicode representation of an IA5String.
0x0003	X509_SUBJECT_ALT_NAME_X400_ADDRESS	The alternative name must be encoded as an `ORAddress` sequence as specified in RFC 2459.

Value	Name	Description
0x0004	X509_SUBJECT_ALT_NAME_DIRECTORY_NAME	The alternative name must be encoded as a `Name` choice as specified in RFC 2459.
0x0005	X509_SUBJECT_ALT_NAME_EDI_PARTY_NAME	The alternative name must be encoded as an `EDIPartyName` sequence as specified in RFC 2459.
0x0006	X509_SUBJECT_ALT_NAME_UNIFORM_RESOURCE_IDENTIFIER	The alternative name must be a Unicode representation of an IA5String.
0x0007	X509_SUBJECT_ALT_NAME_IP_ADDRESS	The alternative name must be an OCTET STRING in "network byte order" as specified in RFC 2459. (network byte order specified in RFC 791).
0x0008	X509_SUBJECT_ALT_NAME_REGISTERED_ID	The alternative name must be encoded as an OBJECT IDENTIFIER as specified in RFC 2459.

Structures 5

The following extensions are defined in this section:

- Section 5.1, "ASN1 Encoded Extension," on page 133
- Section 5.2, "NPKI_Extension," on page 134
- Section 5.3, "Subject Alternative Names Extension," on page 134

5.1 ASN1 Encoded Extension

The following structure allows ASN1 encoded extensions to be specified during the creation of server and user certificates. To add multiple extensions, create a structure for each extension and then link the structures using the `next` field in the structures.

IMPORTANT: Each extension must be a fully ASN.1 encoded extension conforming to RFC 2459.

```
typedef struct NPKI_ASN1_Extensions
{
     NPKI_Extension                   extension;
     struct NPKI_ASN1Extensions       *next;
     NPKI_AltNames                   *altName;    // Array of structures
}

NPKI_ASN1_Extensions;
```

extensions

Contains the values of the ASN1 encoded extension to be encoded into the certificate. See Section 5.2.1, "General Purpose Extension Structure," on page 134.

NOTE: Creating an ASN.1 encoded extension is an advanced operation, requiring detailed knowledge of ASN.1 and X.509 extensions. However, existing ASN.1 encoded extensions may be used without such detailed knowledge. To review an example that uses the parameters to include in an extended key usage extension on the user certificate as it is created, see UserExtendedKeyUsage (../../../samplecode/ncslib_sample/UserExtendedKeyUsage.cpp.html).

`next`

Points to the next node of type NPKI_ASN1_Extensions structure. To add multiple extensions, create a structure for each extension and then link the structures using the `next` field in the structures. The `next` field in last structure in the linked list should be set to null.

altName

Points to an array of NPKI-AltName structures; each element in the array contains one alternative name.

5.2 NPKI_Extension

The NPKI_Extension consists of the general purpose extension structure described in the following section:

5.2.1 General Purpose Extension Structure

The following extension is the genera purpose structure that allows data to be specified for most supported extensions.

```
typedef struct NPKI_Extension
{
     nuint32      flags;
     nuint32      length;      /* length of value */
     nuint8       *value;
}NPKI_Extension;
```

flags

Specifies how the extension is encoded in the certificate. `flag` is composed of both general purpose flags combined with any extension specific flags if necessary. See Section 4.16.1, “General Purpose Extension Flags,” on page 130.

length

Specifies the number of bytes which follow in value. If the extension is not to be encoded in the certificate, flags should be set to PKI_EXTENSION_DONT_INCLUDE, length should be set to 0 and value should be NULL.

value

Points to a byte-array of data.

5.3 Subject Alternative Names Extension

The X.509 subject alternative name extension is used to specify additional identities to be bound to the subject of the certificate (that is, other names that identify the object). See Section 4.3, “General Name Type Extensions,” on page 120.

The subject alternative name extension uses a specific extension structure (NPKI_ExtAltNames or NPKI_AltName) described below:

```
typedef struct      NPKI_ExtAltNames
{
     nuint32             flags;
     nuint16             numberOfNames;
     NPKI_AltNames      *altName;    // Array of structures
}NPKI_ExtAltNames;
```

flags

Specifies how the extension is encoded in the certificate. `flags` is composed of both general purpose flags combined with any subject alternative names extension specific flags. For a description of the general purpose flags, see the Section 4.16.1, "General Purpose Extension Flags," on page 130.

NOTE: There are no subject alternative name extension specific flags defined in this release.

numberOfNames

Specifies the number of elements in the array *altName*.

altName

Points to an array of *NPKI-AltName* structures; each element in the array contains one alternative name. The *NPKI-AltName* structures are described below.

```
typedef struct      NPKI_AltName
{
     nuint16            type;
     nuint16            length;
     nuint8             *value;    }NPKI_AltName;
```

type

Specifies how the subject alternative name is encoded in the certificate. For a description of the types, see the Section 4.3, "General Name Type Extensions," on page 120.

length

Specifies the length (in bytes) of the *value* field (that is, the alternative names).

value

Points to the byte array that contains the alternative name.

NPKI Sample Code

6

See the following Novell Certificate Server samples:

- BackupCA (../../../samplecode/ncslib_sample/BackupCA.cpp.html)
- BackupServerCertificate (../../../samplecode/ncslib_sample/BackupServerCertificate.cpp.html)
- CreateServerCertificate (../../../samplecode/ncslib_sample/CreateServerCert.cpp.html)
- CreateTrustedRoot (../../../samplecode/ncslib_sample/CreateTrustedRoot.cpp.html)
- CreateTrustedRootContainer (../../../samplecode/ncslib_sample/CreateTrustedRootContainer.cpp.html)
- CreateUserCert (../../../samplecode/ncslib_sample/CreateUserCert.cpp.html)
- ExportUserCert (../../../samplecode/ncslib_sample/ExportUserCert.cpp.html)
- FindUserCerts (../../../samplecode/ncslib_sample/FindUserCerts.cpp.html)
- GenerateCSR (../../../samplecode/ncslib_sample/GenerateCSR.cpp.html)
- GetCACert (../../../samplecode/ncslib_sample/GetCACert.cpp.html)
- GetIPandDNSInfo (../../../samplecode/ncslib_sample/GetIPandDNSInfo.cpp.html)
- GetServerInfo (../../../samplecode/ncslib_sample/GetServerInfo.cpp.html)
- GetServerKey (../../../samplecode/ncslib_sample/GetServerKey.cpp.html)
- ImportUserCert (../../../samplecode/ncslib_sample/ImportUserCert.cpp.html)
- LoggingIn (../../../samplecode/ncslib_sample/LoggingIn.cpp.html)
- ReadUserNicknames (../../../samplecode/ncslib_sample/ReadUserNicknames.cpp.html)
- RestoreCA (../../../samplecode/ncslib_sample/RestoreCA.cpp.html)
- RestoreServerCertificate (../../../samplecode/ncslib_sample/RestoreServerCertificate.cpp.html)
- RetrieveServerCertificate (../../../samplecode/ncslib_sample/RetrieveServerCertificate.cpp.html)
- SignCSR (../../../samplecode/ncslib_sample/SignCSR.cpp.html)
- VerifyWithTrustedRoot (../../../samplecode/ncslib_sample/VerifyWithTrustedRoot.cpp.html)
- VersionInfo (../../../samplecode/ncslib_sample/VersionInfo.cpp.html)

Revision History

A

Revision Date	Changes
October 11, 2006	Fixed broken links.
March 1, 2006	• Fixed broken sample code links. • Updated Trademark information.
October 5, 2005	• Transitioned to revised Novell documentation standards.
March 2, 2005	• Fixed broken links and made minor documentation updates to comply with requirements of Novell Forge. • Deprecated Section 4.17, “Subject Alternative Name Types (obsolete, 3/2005),” on page 131 and replaced with the revised Section 4.3, “General Name Type Extensions,” on page 120. • Added Section 4.11, “NPKI_Version Values,” on page 126.
October 6, 2004	• Made technical corrections and fixed broken links
June 9, 2004	• Rolled in the latest version of the 2.72.1 code, which includes all bug fixes for future releases. • Made minor documentation updates.
8 October 2003	Changed name from Novell Certificate Server™ API Version 2 to Novell Certificate Server Library for C Version 2 to clarify differences between Java and C API libraries.
June 2003	• Updated information to include new NDK: Novell Certificate Server Classes for Java API. • Reorganized and edited all chapters.
March 2003	Updated broken Sample Code links.
January 2003	Added as a new component to the Novell® Certificate Server Libraries.