IBM® SecureWay® X.509 Public Key Infrastructure
for Multiplatforms

Programming Guide and Reference

Version 1 Release 1

IBM® SecureWay® X.509 Public Key Infrastructure
for Multiplatforms

Programming Guide and Reference

Version 1 Release 1

Note

Before using this information and the product it supports, read the general information under “Notices” on page 221.

First Edition (October 1999)

This edition applies to IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms, program 5697–F93, version 1 release 1 modification 3, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this book . . . vii
Who should read this book . . . vii
How this book is organized . . . vii
Year 2000 readiness . . . viii
Service and support . . . viii
Conventions . . . viii
Web information . . . viii
Related information . . . viii

Chapter 1. About Secureway X.509 Public Key Infrastructure for Multiplatforms . . . 1
Certificate authority . . . 1
Registration authority . . . 1
End entity . . . 1
Control objects . . . 1
Object store . . . 2

Chapter 2. PKIX certificates . . . 3
Certificate life cycle . . . 3
Setting fields in a certificate request . . . 4
Certificate extensions . . . 5
  Extension identifiers and values . . . 5
  Parameter format . . . 7
  Writing a certificate extension program . . . 8

Chapter 3. Writing applications for server components . . . 9
Initializing servers . . . 9
Registering API callbacks . . . 10

Chapter 4. Writing applications for the end entity . . . 13
Creating and submitting certificate requests . . . 13
  Creating a certificate request . . . 13
  Submitting a certificate request to the registration authority . . . 14
  Creating a certificate authority certificate request . . . 15
Creating a revocation request . . . 15

Chapter 5. Writing applications for the registration authority . . . 17
Pre-registering end entities . . . 17
Processing certificate requests . . . 17
Processing revocation requests . . . 18

Chapter 6. Writing applications for the certificate authority . . . 21
Approving and rejecting certificate requests . . . 21
Revoking certificates . . . 21
  Approving a revocation request . . . 21
  Rejecting a revocation request . . . 22
  Creating a certificate revocation list (CRL) . . . 22

Chapter 7. Application Programming Interfaces . . . 25
JNH_add_certreq_extension . . . 31
JNH_authorize_registration . . . 34
JNH_authorize_revocation . . . 35
JNH_BootStrap . . . 36

JNH_CA_list_RAs . . . 37
JNH_CA_list_signing_keys . . . 38
JNH_CA_nonpkix_user_check . . . 39
JNH_CA_nonpkix_user_done . . . 40
JNH_CA_nonpkix_user_get_cert . . . 41
JNH_CA_preregister_nonpkix_user . . . 42
JNH_CA_register_nonpkix_user . . . 43
JNH_CA_write_info . . . 44
JNH_cancel_revreq . . . 45
JNH_cert_inquire_endDate . . . 46
JNH_cert_inquire_issuer . . . 48
JNH_cert_inquire_keyUsage . . . 49
JNH_cert_inquire_serialNumber . . . 51
JNH_cert_inquire_startDate . . . 52
JNH_cert_inquire_subject . . . 54
JNH_cleanup_BootStrap . . . 55
JNH_confirm_msg . . . 56
JNH_create_BootStrap . . . 57
JNH_create_certificate . . . 58
JNH_create_CRL . . . 59
JNH_create_enrollment_request . . . 60
JNH_create_revreq . . . 61
JNH_create_revreq_from_certificate . . . 62
JNH_delete_object . . . 63
JNH_enroll . . . 64
JNH_enroll_RA . . . 65
JNH_export_credential . . . 66
JNH_get_CA_info . . . 68
JNH_get_error . . . 69
JNH_get_fingerprint . . . 70
JNH_get_IniMyName . . . 71
JNH_get_IniLdapAuthName . . . 72
JNH_get_IniLdapAuthPwd . . . 73
JNH_get_IniLdapServer . . . 74
JNH_get_IniTcpHost . . . 75
JNH_get_IniTcpPort . . . 76
JNH_get_self_serial_number . . . 77
JNH_get_self_subjectKeyInfo . . . 78
JNH_get_object_state . . . 79
JNH_GetStatus . . . 80
JNH_INI_deleteKey . . . 81
JNH_INI_deleteSection . . . 82
JNH_INI_Initialize . . . 83
JNH_INI_readKeys . . . 84
JNH_INI_readSections . . . 86
JNH_INI_readString . . . 87
JNH_INI_writeFile . . . 89
JNH_INI_writeString . . . 90
JNH_initialize_UI . . . 91
JNH_inquire_certreq_basicConstraints . . . 92
JNH_inquire_certreq_enddate . . . 94
JNH_inquire_certreq_issuer . . . 95
JNH_inquire_certreq_keyUsage . . . 96
JNH_inquire_certreq_privkey_EE . . . 98
JNH_inquire_certreq_serialnumber . . . 99
JNH_inquire_certreq_startdate . . . 100

JNH_inquire_certreq_status 101
JNH_inquire_certreq_subject 102
JNH_inquire_certreq_subjectkey_algorithm 103
JNH_inquire_revreq_certIssuer 104
JNH_inquire_revreq_certserialnumber 105
JNH_inquire_revreq_certserialnumbers 106
JNH_inquire_revreq_hold_instruction_code 108
JNH_inquire_revreq_invalidityDate 109
JNH_inquire_revreq_reason 111
JNH_inquire_revreq_requests 113
JNH_Keypair_Selected 114
JNH_keystore_inquire_cert 115
JNH_List_Existing_Keypairs. 117
JNH_list_objects 118
JNH_list_SC_certs 119
JNH_list_surrogates 121
JNH_modify_certreq_extension 122
JNH_new_revreq 124
JNH_pkcs12scExportFile 125
JNH_pkcs12UserExportFile 130
JNH_pkcs12ImportFile 133
JNH_preregister_crosscert 136
JNH_preregister_user 137
JNH_publish_certificate 139
JNH_publish_CRL 140
JNH_RA_nonpkix_create_revreq 141
JNH_RA_nonpkix_request_revocation 142
JNH_RA_nonpkix_user_check. 143
JNH_RA_nonpkix_user_done 144
JNH_RA_nonpkix_user_get_cert 145
JNH_RA_preregister_crosscert 146
JNH_RA_preregister_nonpkix_user 148
JNH_RA_preregister_user 149
JNH_RA_register_nonpkix_user 151
JNH_register_callbacks 152
JNH_register_user 154
JNH_reject_registration 155
JNH_reject_revocation 156
JNH_release_object 158
JNH_release_octetString 159
JNH_remove_certreq_extension 160
JNH_request_CRL 161
JNH_request_revocation 162
JNH_reserve_object 163
JNH_revoke_certificate 164
JNH_save_object 165
JNH_server_login_pwd 166
JNH_Set_CCert_VerificationKey 167
JNH_set_certreq_basicConstraints 168
JNH_set_certreq_endDate 170
JNH_set_certreq_issuer 172
JNH_set_certreq_keyUsage 173
JNH_set_certreq_privkey_EE 175
JNH_set_certreq_startDate 177
JNH_set_certreq_subject. 179
JNH_set_certreq_subjectKeyInfo 180

JNH_set_IniLdapAuthName . . . 181
JNH_set_IniLdapAuthPwd . . . 182
JNH_set_IniLdapServer . . . 183
JNH_set_IniTcpHost . . . 184
JNH_set_IniTcpPort . . . 185
JNH_set_MyName . . . 186
JNH_set_RA_URL . . . 187
JNH_set_revreq_certIssuer . . . 189
JNH_set_revreq_certserialnumber . . . 191
JNH_set_revreq_hold_instruction_code . . . 193
JNH_set_revreq_invalidityDate . . . 194
JNH_set_revreq_reason . . . 196
JNH_set_server_location . . . 198
JNH_shutdown_UI . . . 199
JNH_start_server . . . 200
JNH_stop_server . . . 201
JNH_store_RA_URL . . . 202
JNH_subject_submit_crosscert . . . 203
JNH_validate_registration . . . 204

Appendix A. Using the ASN.1 class library . . . 205
ASN.1 data types . . . 205
PKIX interface data types . . . 206
BER and DER encoding . . . 206
ASN.1 class library hierarchy . . . 207
Abstract classes . . . 207
ASN.1 header files . . . 208

Appendix B. PKIX programming model . . . 209
Introduction . . . 209
Programming the CA and RA bootstraps . . . 211
Sample bootstrap C++ sequence calls . . . 212
Enrolling the RA with the CA . . . 213
Sample RA enrollment code . . . 214
  Initiating the RA enrollment request . . . 214
  CA processing of the RA enrollment request . . . 216
Programming to the end entity . . . 216
Sample EE certificate life cycle code . . . 217
  Performing user preregistration at the RA . . . 217
  Requesting a certificate . . . 218
Programming to the registration authority . . . 219
Programming to the certificate authority . . . 219
Header and library files . . . 220

Notices . . . 221
Trademarks . . . 223

Glossary . . . 225

Index . . . 235

About this book

This book provides information for programmers developing applications using the IBM® SecureWay® X.509 Public Key Infrastructure for Multiplatforms, hereafter referred to as PKIX. The topics discussed include:

- An overview of the product.
- Guidelines for developing applications for the PKIX certificate authority (CA) and registration authority (RA), and end entitiy, or client, components.
- Public Key Infrastructure (PKI) for X.509 certificates, certificate extensions, and certificate life cycle.
- Application program interface (API) that enable you to access and manipulate PKI functions.
- Abstract Syntax Notation One (ASN.1) data types

Who should read this book

This book is for application developers familiar with C, C++, and Java™ programming languages. You should be knowledgeable about the following concepts:

- Internet communications protocols
- Public key infrastructure technology, including Directory schemas, the X.509 version 3 standard, and Lightweight Directory Access Protocol (LDAP)
- Cryptography

How this book is organized

This book is organized as follows:

- **Chapter 1. About Secureway X.509 Public Key Infrastructure for Multiplatforms** gives a brief description of PKIX and some its key components.
- **Chapter 2. PKIX certificates** discusses certificates and certificate life cycle, certificate extensions and how to write one, and parameter formats.
- **Chapter 3. Writing applications for server components** explains how to initialize the server components and register API callbacks.
- **Chapter 4. Writing applications for the end entity** discusses creating and submitting certificate requests for the end entity. A discussion on creating revocation requests is also included.
- **Chapter 5. Writing applications for the registration authority** describes how to write some common tasks for the RA, which include preregistering the end entity, and processing revocation requests.
- **Chapter 6. Writing applications for the certificate authority** shows to how approve or reject certificate requests, process revocations, and create certificate revocation lists.
- **Chapter 7. Application Programming Interfaces** contains reference information on the APIs available along with coding examples to assist you during development.
- **Appendix A. Using the ASN.1 class library** explains the Abstract Syntax Notation One (ASN.1) coding notation and data types, and shows examples of how to use this notation.
- **Appendix B. PKIX programming model** describes an overall approach, with narratives and coding examples, to developing a PKIX application.

A glossary of terms and an index are also provided.

Year 2000 readiness

These products are Year 2000 ready. When used in accordance with their associated documentation, they are capable of correctly processing, providing, and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with the products properly exchange accurate date data with them.

Service and support

Contact IBM for service and support for all the products included in the IBM SecureWay FirstSecure offering. Some of these products may refer to non-IBM support. If you obtain these products as part of the FirstSecure offering, contact IBM for service and support.

Conventions

The following conventions are used in this book:

Convention	Meaning
`monospace`	Syntax, sample code.
italic	Variable values the user must supply.

Web information

Information about updates to IBM SecureWay FirstSecure products is available at the following Web address:

```
http://www.ibm.com/software/security/firstsecure/library
```

Related information

The *IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms Installation Guide* contains information about the PKIX architecture, installation, and configuration.

Chapter 1. About Secureway X.509 Public Key Infrastructure for Multiplatforms

IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms (PKIX) provides a set of software that enables you to create, manage, store, and revoke certificates. Its support for Public Key Infrastructure for X.509 version 3 standard and Common Data Security Architecture (CDSA) allows for flexible application development and vendor interoperability. PKIX provides the following components:

- Certificate authority (CA)
- Registration authority (RA)
- Support for a client system, also referred to as an end entity (EE)
- Control objects
- Object stores

Certificate authority

The certificate authority (CA) is a server application trusted by one or more users to create, manage, and revoke certificates.

The CA is responsible for the following:

- Enforcing security policies (such as a certificate validity period or key revocation policy)
- Creating certificates
- Revoking certificates
- Maintaining lists of trusted RAs and cross-certified CAs

Registration authority

The registration authority (RA) is a server application entity given responsibility for performing some of the administrative tasks necessary in the registration of subjects, including the following tasks:

- Confirming the subject's identity
- Validating that the subject is entitled to have the attributes requested in a certificate
- Verifying that the subject has possession of the private key associated with the public key requested for a certificate
- Approving certificate creation and revocation requests

End entity

The end entity (EE) initiates certificate creation and revocation requests and maintains the private keys and certificates issued by PKIX for a user.

Control objects

PKIX uses two objects to control the processes of creating and revoking certificates: the certificate request and the revocation request. Each of these objects contain information needed by the RA or CA to create or revoke certificates. The objects are primarily created at the EE (although the RA and CA can also create them) and sent from the EE to the RA to the CA and back again. The RA and CA can modify these requests to add their own constraints (such as a specific length of time for certificate validity). The CA converts a certificate request into a certificate and one or more revocation requests into a certificate revocation list (CRL).

Once a certificate request has been issued, a copy of that certificate is placed in the certificate store, where the CA can retrieve information about the certificate during its lifetime. The certificate store contains copies of each certificate created by the CA, indexed by serial number. These copies can be queried using the JNH_inquire_certreq_*xxx* set of APIs. Issued certificates are also stored in the requesting EE's key store and can be accessed using the JNH_cert_*xxx* set of APIs. Certificates are also published to the LDAP directory.

Object store

Each PKIX component has an object store used to track certificate request and revocation request transactions. Objects in this store are either active control objects (certificates, requests, CRLs) or surrogates. A surrogate is a place holder for a control object that has passed through the object store and is expected to return. Any information or state of the control object can be stored in the surrogate until the control object's return. When an active control object returns to the object store, the surrogate is deleted.

Objects in the object store are flagged with a state, such as “Active” or “Revreq Submitted”. As objects move into and out of the object store their states change, reflecting any action that has occurred. For example, while a certificate request is in the EE's object store, it is flagged as “Active”. When it is sent onto the RA, however, the remaining surrogate object is flagged “Submitted”. To find out the state of a request, perform an “and” (&) operation of the status and 0xffff0000, then compare the results with the states defined in the ObjStates.h file.

The object store is indexed using “request identifiers”, ASN.1-encoded integers. Each object is assigned a request identifier, which is used in future API calls on the object. The request identifier corresponds with the “reqId” parameter of the PKIX APIs.

Note: After a surrogate object is deleted, its request identifier is reused for the next surrogate object created in the object store.

Chapter 2. PKIX certificates

This chapter provides information about PKIX certificates. It includes the certificate life cycle and information about certificate fields and extensions.

Certificate life cycle

The certificate life cycle consists of two components: certificate creation and certificate revocation.

Certificate creation:

1. The RA creates a preregistration record for the EE.

   The preregistration record contains a transaction identifier for the EE. This record must be communicated to the EE through electonic mail or some other means.

   Optionally, the RA, using delayed authentication, can create a preregistration record without specifying a subject name. An EE can then fill in the name and use the record to register. However, the RA will have to validate the user.

   To create a certificate request that uses delayed authentication, the RA checks the **Authenticate User** box on the GUI.
2. The EE uses the preregistration record to request a certificate.

   The EE creates a certificate request and submits it to the RA.
3. The RA approves the request and sends it to the CA.

   The RA reviews the request to determine if all necessary security criteria are met and that all the required fields are completed and accurate. It is the responsibility of the RA to verify the identity of an applicant prior to approving a registration request.
4. The CA approves the certificate request and creates the certificate.
5. The RA periodically polls the CA to determine if the certificate request has been processed. If the CA has approved the request, it will return the issued certificate to the RA as part of the poll response. The CA converts the certificate request in its object store to an issued certificate.
6. The EE periodically polls the RA to determine if the RA has completed processing the certificate request. If the RA has received the issued certificate from the CA, the RA returns the issued certificate to the EE as part of the poll response.
7. After the EE has successfully received the certificate from the RA and successfully stored it in a smart card or exported it, the EE sends a confirmation message to the RA to indicate that it has successfully received the certificate.
8. When the RA receives the confirmation message from the EE, the RA posts the issued certificate to an LDAP directory, making it publicly available.

Certificate revocation:

1. The EE creates a revocation request and sends it to the RA.

   Certificates can be revoked for a number of reasons, such as the following:
   - The private key has been compromised.
   - The CA has been compromised.
   - The certificate holder’s affiliation has changed.
   - The certificate is never used.
2. The RA approves the revocation request and sends it to the CA.

The RA reviews the request to ensure the identity of the requester and the reason for the revocation.

3. The CA approves the request, revokes the certificate, and adds it to a certificate revocation list (CRL). The CA periodically sends updated certificate revocation lists to the RA. The RA will then publish the CRL to the LDAP directory.

   Note: The revocation does not take affect until the RA publishes the CRL.

Setting fields in a certificate request

Certificates contain several fields, which provide information about the subject of the certificate, as well as the privileges that accompany the certificate. These fields include start date and expiration date, subject name, certificate issuer, and the subject's public key. You can add, modify, and remove these fields using the PKIX APIs.

Note: Any changes to the fields must either be made before you submit the certificate request or be approved by the RA or CA.

For example, you can set the validity period for a certificate with calls to JNH_set_certreq_startdate and JNH_set_certreq_enddate. (If you do not specify a validity period, the period will begin when the certificate is issued and continue for the default lifetime specified in the .ini files.)

Note: Before you run any JNH_set_*xxx* routine against an object, you must first lock that object with a call to JNH_reserve_object.

The following code shows an EE setting the starting and ending dates of the validity period for a certificate:

```
#include <jonah.h>
uint32 id = 1;           // Request Id
uint32 status;

utcDateTime sdStruct;     //Start date
utcDateTime edStruct;     //End date

sdStruct.year = 1999;     //Sample start date information
sdStruct.month = 6;
sdStruct.day  = 23;
sdStruct.hour = 0;
sdStruct.min = 0;
sdStruct.sec = 0;
sdStruct.msec  = 0;

edStruct.year = 1999;     //Sample end date information
edStruct.month = 8;
edStruct.day  = 23;
edStruct.hour = 0;
edStruct.min = 0;
edStruct.sec = 0;
edStruct.msec  = 0;


if (status = JNH_reserve_object(id))
{
   printf("Error calling reserve object with returned status = %d\n", status);
}

if (status = JNH_set_certreq_startDate(id, sdStruct))
{
   printf("Error calling JNH_set_certreq_startDate with returned status = %d\n", status);
}

if (status = JNH_set_certreq_endDate(id, edStruct))
{
   printf("Error calling JNH_set_certreq_endDate with returned status = %d\n", status);
}
```

For information about other APIs that can be used to modify certificate fields, see “Chapter 7. Application Programming Interfaces” on page 25.

Certificate extensions

Certificate extensions are attributes that are associated with the certificate. There are three types of extensions:

standard extensions
: Extensions such as Key Usage and Subject Alternate Name, which are needed for the SSL and S/MIME protocols.

common extensions
: Extensions known to PKIX, such as host identity mapping, which associates the subject of a certificate with a corresponding identity on a host system.

private extensions
: Extensions that are user-defined and whose purpose is specific to a particular security implementation.

The EE requests extensions, but the CA (or an RA acting on the CA’s behalf) must validate those requests. The process is as follows:

1. The EE, through the GUI or an API call, requests an extension and supplies information for the extension, including a unique extension identifier and value, and specifies whether or not the extension is critical.

   If a system encounters an extension it does not recognize, such as a private extension, the system accepts the extension as long as it is not marked critical.
2. The extension request and information become part of a certificate request in the object store. From there the request is sent to the RA.
3. The RA or CA, while processing the certificate request, validates the extension against the organization’s certificate policy and modifies or overrides the extension request. If the extension is validated, the CA certifies it.

For more detailed information about certificate extensions, including definitions of the different extensions, see the IETF’s RFC 2459 at http://www.ietf.org.

Extension identifiers and values

Extension identifiers are unique strings followed by a named OID specific to the extension. Extension values define the purpose of an extension (such as Key Usage). The following table shows the ASN.1 notations of different extension identifiers and values:

Extension	Identifier	Value

Key Usage	2.5.29.15	`KeyUsage ::= BIT STRING(`
		`     digitalSignature     (0),`
		`     nonRepudiation       (1),`
		`     keyEncipherment      (2),`
		`     dataEncipherment     (3),`
		`     keyAgreement         (4),`
		`     keyCertSign          (5),`
		`     cRLSign              (6),`
		`     encipherOnly         (7),`
		`     decipherOnly         (8)`

Subject Alternative Name	2.5.29.17	`SubjectAltName ::= GeneralNames`
		
		`GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName`
		
		`GeneralName := CHOICE {`
		`  otherName             [0]     OtherName,`
		`  rfc822Name            [1]     IA5String,`
		`  dNSName               [2]     IA5String,`
		`  x400Address           [3]     ORAddress,`
		`  directoryName         [4]     Name,`
		`  ediPartyName          [5]     EDIPartyName,`
		`  uniformResourceIdentifier  [6] IA5String,`
		`  iPAddress             [7]     OCTET STRING,`
		`  registerID           [8]   OBJECT IDENTIFIER}`
		
		`EDIPartyName ::= SEQUENCE {`
		`   nameAssigner  [0]   DirectoryString  OPTIONAL,`
		`   partyName     [1]   DirectoryString }`

Host identity mapping	1.3.18.0.2.18.1	`HostIdMappings ::= SET OF Mapping` where Mapping is defined below.
Private	user-provided numeric OID	`PrivateValue ::= OCTET STRING`

Notes:

1. For non-CA certificate requests that use RSA keys, the following key usages are available:
   - digitalSignature
   - nonRepudiation
   - keyEncipherment
   - dataEncipherment

   CA certificates may contain all of the key usages that are specified for the non-CA certificate key usages. However, a CA key cannot be used for both certificate or CRL signing and enciphering.
2. PKIX supports the use of IPAddress and e-mail address (for the rfc822Name field) for the Subject Alternative Name extension.
3. The value for the Host identity mapping extension allows one certificate extension to contain multiple Host-Id mappings. Because of this, you can associate a certificate holder with multiple identities on multiple hosts, as defined below:

```
HostIdMappings ::= SET OF Mapping, and
Mapping ::= SEQUENCE { hostName   IMPLICIT[1]  IA5String,
                      subjectId  IMPLICIT[2]  IA5String,
                      id_pop IdProof }, and
IdProof ::= SEQUENCE { secret                OCTET STRING,
                        encryptionAlgorithm   AlgorithmIdentifier }
```

AlgorithmIdentifier is defined in the x509.h header file.
4. The private extension must not be critical.
5. Only one extension of each type can be added to each certificate. A host identity mapping can be included in the Host Identity Mapping extension for each unique host name, and a private extension can be added for each unique extension identifier.

Parameter format

This section discusses the parameter format for distinguished names and object identifiers.

Distinguished names

Among the most common features of a certificate are distinguished names, or DNs. A DN consists of a sequence of relative distinguished names, or RDNs, and an RDN consists of a set of attribute value assertions (AVAs) which are essentially name/value pairs. All DNs are represented in the APIs in a string format known as the OSF syntax.

In the OSF syntax, an AVA is represented by a standard keyword for a directory attribute, an equal sign, and then the value as a UTF-8 string. For example, C=US represents “Country Name” is “US” (for the United States of America). The common attributes supported within the DNs are listed in Table 1.

UTF-8 is a string format representing Unicodes in which all IA5 characters (IA5 is an international character set very similar to US ASCII) have the same representation as they do in IA5, while other characters consist of multiple bytes, all of which have values in the range 128–253. A more detailed definition of UTF-8 can be found in the IETF’s RFC 2279.

In the OSF syntax, an RDN is represented by a solidus (“/”) character followed by a series of AVA representations separated by commas. A DN is represented by a series of RDNs, with the most general RDN coming first. In a typical example, the first RDN will contain the “Country Name” attribute and no others, while the last RDN will contain the “Common Name” attribute.

Table 1. Common attributes supported within Distinguished Names

Parameter	Description	Comments
C	Country Name	Two ASCII upper-case letters
Common Name	User’s full name	
O	Organization Name	
OU	Organizational Unit	There may up to four of these.
ST	State or province name	
L	Locality name	
MAIL	E-mail address	Must be in IA5, in *user@domain* format
PC	Postal code	
T	Title	

Object identifiers

Most object identifiers (OIDs) are represented in this API in “numeric OID dot-format”. This format consists of a sequence of non-negative integers separated by periods. The first number in the sequence should always be 0, 1, or 2.

Some object identifier parameters are represented in the standard ASN.1 binary representation for OIDs. In this format, any number in the dot-format which is between 0–127 inclusive is represented by a single byte with that value. Any number in that format which is between 128–16383 inclusive is represented by two bytes, the first of which is that number divided by 128 added to 128, and the second of which is that number modulo 128; while any number in that format between 16384–2097151 inclusive is represented by three bytes. The first byte is that number divided by 16384 added to 128. The second byte is that number divided by 128 modulo 128 added to 128. The third byte is that number modulo 128. The general form is that a number between 2**7n and 2**7(n+1) is encoded as n+1 bytes, with each (right-justified and zero-filled) group of 7 bits in a separate byte, with the high-order byte first and with all bytes other than the last one having the high-order bit set.

Writing a certificate extension program

The following code shows how to add a certificate extension to the request and modify the extension.

Note: Before you add, modify, or remove an extension from a certificate, you need to determine the corresponding request identifier from the EE’s object store.

```
#include <x509.h>
#include <Jonah.h>

octetString value;
int critical= 0
xtype = SUBJECT_ALT_NAME;
utf8String extId = "2.5.29.17";
value.data = (unsigned char *)strdup("127.0.0.1");
value.length = strlen((char *)value.data);
uint32 rc = JNH_add_certreq_extension(xtype, reqId, extId, value, critical);
JNH_release_octetString(value);  // Release octetString associated with value

octetString newValue = ;
uint32 new_critical = ...;
status = JNH_modify_certreq_extension ( extn_type,
                               req_id,
                               extn_id,
                               new_value,
                               new_critical);
```

Chapter 3. Writing applications for server components

This chapter provides information about writing applications that run on all three server components (CA, RA, and EE). It includes information about initializing servers and registering callbacks. Refer to “Appendix B. PKIX programming model” on page 209 for additional information on developing applications.

Initializing servers

Each PKIX server must initialize itself using JNH_start_server. This initialization connects the server to the appropriate object stores and restricts functions on certificate requests and revocation requests. When starting an RA or CA server, you must log in to the smart card by using JNH_server_login_pwd.

Note: JNH_start_server passes one parameter, serverType, which specifies the type of server being initialized. Optionally, you can specify a second parameter indicating the path to the .ini file. In the following code samples, the servers are specified with the following parameters, which are literals defined in jonah.h:

EE server	svrType_EE
RA server	svrType_RA
CA server	svrType_CA

The following code is an example of initializing an EE server:

```
void start_ee ()
{
JNH_start_server(svrType_EE, (char *) "EE_file.ini");
}
```

The following code is an example of initializing an RA server:

```
uint32 start_RA()
{
uint32 status;
utf8String passWord[] = "RApass";  //Password created when initializing RA

JNH_start_server(svrType_RA, (char *) "RA_file.ini");
status = JNH_server_login_pwd(passWord);

return status;
}
```

The following code is an example of initializing a CA server:

```
uint32 start_CA()
{
uint32 status;
utf8String passWord[] = "CApass";  //Password created when initializing CA

if (status = JNH_start_server(svrType_CA, (char *) "CA_file.ini") return status;
status = JNH_server_login_pwd(passWord);

return status;
}
```

Registering API callbacks

To register API callbacks in the PKIX servers, you use a call to JNH_register_callbacks. JNH_register_callbacks allows applications to react to events occurring on the PKIX servers.

JNH_register_callbacks() uses two arguments: a pointer to a user-defined JNH_Notify function, and a pointer to a user-defined JNH_Display function. These APIs allow you to define how your application reacts to events and debug messages. JNH_register_callbacks() stores the addresses of these functions, which are invoked whenever an event occurs or there is a debug message to display on the screen.

JNH_Notify returns the state of an event, using three arguments: request id, requester name, and status of a certificate. JNH_Notify uses the status to determine the state of a certificate.

JNH_Display displays error and debug messages by using two arguments: message type and debug message. JNH_Display uses the message type to sort, log, or print out the messages. To determine a message type, see the Jonah.h file.

The following code shows how to register callbacks:

```
//display function
void display(uint32 type, const utf8String message)
{
      switch(type) {
      case DISPLAY_STATUSBAR:
        fprintf(stderr, "bar -> ");
        break;
      case DISPLAY_LOGERROR:
        fprintf(stderr, "log error -> ");
        break;
     case DISPLAY_LOGINFO:
        fprintf(stderr, "log info -> ");
        break;
      case DISPLAY_LOGDEBUG:
        fprintf(stderr, "log debug -> ");
        break;
     case DISPLAY_URGENTERROR:
        fprintf(stderr, "urg error -> ");
         break;
     case DISPLAY_URGENTINFO:
        fprintf(stderr, "urg info -> ");
        break;
     case DISPLAY_URGENTDEBUG:
        fprintf(stderr, "urg debug -> ");
        break;
      default:
        fprintf(stderr, "unk type -> ");
         break;
   }
      fprintf(stderr, "%s\n", message);
}

void notify(uint32 id, const utf8String name, uint32 status)
{
      uint32 foo;

      foo = status & 0xffff0000;
      if (foo == ObjStEECertReqActive)
{
          fprintf(stderr, "createCertReq: NAME=%s\tID=%d\tSTATUS=%x\n", name,
                 id, status);
   }

}

void main(int argc, char *argv[ ])
{

      if (status = JNH_start_server(svrType_EE, (char *)"EE_file.ini")) return status;
      if (status = JNH_register_callbacks(&notify, &display)) return status;

}
```

Chapter 4. Writing applications for the end entity

This chapter provides information for developing applications that run on the EE component. It includes sample programs for the different EE tasks and discusses special considerations for EE application development.

Creating and submitting certificate requests

The EE uses two APIs to create and submit a certificate request: JNH_preregister_user and JNH_register_user.

Creating a certificate request

The EE creates a certificate request through a call to JNH_preregister_user, which retrieves information from the preregistration request that was created by the RA. After a request is created, it is flagged “Active” and stored in the EE’s object store until it is submitted to the RA.

The following code is an example of an EE creating a certificate request:

```
// This file name represents a file where the preregistration record is found
char filename[] = "c:\temp\test.reg";

char tempFile[80];
FILE *fp = NULL;
int size = 0;
unsigned char *prereginfo = NULL;
utf8String password = "pass";
   ⋮

if (status = JNH_start_server (svrType_EE, (char *) "EE_file.ini")) return status;

//Example of reading the preregistration record created by the RA out of a file
if ((fp = fopen(filename, "rb")) == NULL) {
{
      fprintf(stderr, "Can not open %s\n", filename);
        exit (-1);
}
// Find the length of the file to use allocate memory
fseek(fp, 0L, SEEK_END);
size = ftell(fp);
rewind(fp);
if (size < 2)
{
      fprintf(stderr, "Message too small\n");
      exit (-1);
}
if ((fp = (unsigned char *) malloc(size)) == NULL)
{
      fprintf(stderr, "Out of memory\n");
      exit (-1);
}
memset(prereginfo, 0, size);
if (fread(prereginfo, size, 1, fp) != 1)
{
      fprintf(stderr, "Read Error!\n");
      exit (-1);
}
fclose(fp);

/* Call JNH_pregregister_user with the preregistration record information.
**  The second  parameter is for a password. It returns a RequestID
**  for use in continuing with the certificate lifecycle.
```

```
*/
if((status = JNH_preregister_user(prereginfo, password, &id)))
{
      printf("prereg status = %d\n", status);
      exit (-1);
}
   ⋮
```

Submitting a certificate request to the registration authority

Once the certificate request is created and any modifications to the certificate fields complete, the EE submits the request through a call to JNH_register_user. However, before submitting the certificate, the EE must perform the following steps:

1. Use JNH_reserve_object to reserve the certificate request in the object store.
2. Set the private key with JNH_set_certreq_privkey_EE.
3. Save the changes to the certificate request with JNH_save_object.
4. Export the information to the smart card, using JNH_export_credential.

Once the certificate request is submitted to the RA, it is stored in the RA's object store and is flagged “Active”. The corresponding surrogate object in the EE's object store is flagged “Submitted”.

The following code is an example of an EE submitting a certificate request:

```
//Get object store ID for your ACTIVE certificate
uint32 status;
uint32 id = 1;     //Request identifier
   ⋮

if((status = JNH_reserve_object(id)))
{
      printf("reserve returned %d\n", status);
      exit (-1);      // or return to calling routine
}

if ((status = JNH_set_certreq_privkey_EE(id, (utf8String) "id-dsa", 512)))
{
      printf("set ee generated status = %d", status);
      exit (-1);      // or return to calling routine
}

if ((status = JNH_save_object(id)))
{
      printf("save returned %d\n", status);
      return status;
}

/* Get EE virtual smart card password and device information
 * Device is one of:
 *
 *   "VSC:fileName"
 *   "TOK:readerName"
 *
 * where VSC indicates the virtual smart card and TOK indicates
 * a real smart-card.
 */

char ECpwd[] = "ee1234";     //EE smart card password
char tokenFile[] = "VSC:c:\token.fil";

if ((status = JNH_export_credential(id, (utf8String) ECpwd,(utf8String) tokenFile)))
{
      printf("export credential status = %d\n", status);
      return status;
}

if ((status = JNH_register_user(id)))
{
      printf("register user status = %d\n", status);
      return status;
}
   ⋮
```

Note: The RA server must be running to receive this request.

Creating a certificate authority certificate request

To create a certificate request for a CA, specify in the certificate request a basic constraint that indicates that the certificate's subject is a CA. You can do this through a call to JNH_set_certreq_basicConstraints.

The following code is an example of this:

```
//certificate type is CA
uint32 status;
uint32 id = 1;              // Request identifier
bool isCaCert = true;
int maxPathLen = 150;
   ⋮

if (isCaCert)
      if ((status = JNH_set_certreq_basicConstraints(id, isCaCert, maxPathLen,0)))
      {
printf("Error setting basicConstraints with status = %d", status);
return status;
      }
   ⋮
```

Creating a revocation request

If an EE believes that a certificate has been compromised or if the certificate is no longer needed, the EE can request that the certificate be revoked. The EE creates a revocation request with a call to JNH_create_revreq and submits it to the RA with a call to JNH_request_revocation.

Once the request is submitted to the RA, it is stored in the RA's object store and flagged "Revreq Active". The corresponding surrogate object in the EE's object store is flagged "Revreq Submitted".

The following code is an example of an EE creating a revocation request:

```
uint32 rc;
uint32 reqid = 3;      // Request identifier
   ⋮

rc = JNH_create_revreq (reqid);
   ⋮
```

The following code is an example of an EE submitting the revocation request to the RA:

```
uint32 reqid = 3;      // Request identifier
   ⋮

rc = JNH_request_revocation (reqid);
```

⋮

Note: The RA server must be running to receive this request.

Chapter 5. Writing applications for the registration authority

This chapter provides information for developing applications that run on the RA component. It includes sample programs for the different RA tasks and discusses special considerations for RA application development.

Pre-registering end entities

The RA preregisters end entities with JNH_RA_preregister_user, which creates a preregistration record that is stored in the RA's object store. The RA forwards this record to the EE through a secure channel (such as electronic mail), and the EE uses it to create and submit a certificate request. The surrogate for this preregistration record in the RA's object store is flagged “Preregistered”.

The following code is an example of an RA preregistering an EE:

```
char name[] = "user";        // name of user to preregister
char pwd[] = "pass";         // password for user
utf8String CAName = NULL;    // Certificate Authority name - from ini file
uint32 status;               // return value
int expire_secs = 120;       // number in seconds to complete registration
utf8String buf;              // buffer to hold Preregistration record returned
                                // from JNH_RA_preregister_user
   ⋮


   if (status = JNHstart_server (svrType_RA, (char *) "RA_file.ini")) return status;

// This gets the name of the CA from the INI file
if (status = JNH_INI_readString((unsigned char *const)"General",
      (unsigned char *const)"Issuer1", (unsigned char **) &CAName,
      (unsigned char *const)""))
{
      printf("PreregisterUsers: Cannot call JNH_INI_readString() with status =
            %d.\n", status);
      return status;
}
if ((status = JNH_RA_preregister_user(CAName, (utf8String) name, expire_secs,
      (utf8String) pwd, &buf)))
{
      printf("PreregisterUsers: Cannot call JNH_RA_preregister_user() with status =
             %d.\n", status);
     return status;
}
   ⋮
```

Processing certificate requests

When the EE sends a certificate request to the RA, the RA's notify function is called to notify the RA of the new request. The RA can then either authorize or reject the request. If the RA authorizes the request (with a call to JNH_authorize_registration), it is forwarded to the CA and stored as an “Active” object in the CA's object store. The surrogate object in the RA's object store is flagged “Waiting for CA”.

The following code shows an RA authorizing a certificate request:

```
uint32 id = 1;       //Request identifier
uint32 keyid = 0;    //Key ID
uint32 status;
```

```
   ⋮
status = JNH_authorize_registration(id, keyid);
return status;
   ⋮
```

Note: The CA server must be running to receive this request.

If the RA rejects a certificate request (with a call to JNH_reject_registration), either because it is not complete or is invalid, the request is returned to the EE with a reason for the rejection. The object in the RA's object store that corresponds to the certificate request is flagged "Rejected" and is removed from the object store when the EE receives the request.

The following code is an example an RA rejecting a certificate request:

```
uint32 id = 1;                      //Request identifier
uint32 status;
utf8String reason = "Invalid data";  //Reason for rejecting request
   ⋮
status = JNH_reject_registration(id, reason);
return status;
   ⋮
```

Processing revocation requests

When the EE sends the RA a request to revoke a certificate, the RA's notify function is called to notify the RA of the new request. The RA authenticates the identity of the requestor and can then authorize the revocation with JNH_authorize_revocation or reject it with JNH_reject_revocation.

If the RA authorizes the revocation request, the request is forwarded to the CA and stored in the CA's object store with a state of "Revreq Active". The surrogate object in the RA's object store is flagged "Revreq Waiting for CA". The following code is an example of an RA authorizing a revocation request:

```
uint32 id = 1;       //Request identifier
uint32 status;
   ⋮
status = JNH_authorize_revocation(id);
return status;

   ⋮
```

Note: The CA server must be running to receive this request.

If the RA rejects the revocation request, a message is sent back to the EE with the reason for the rejection. The following code is an example of this:

```
uint32 id = 1;                         //Request identifier
uint32 status;
utf8String reason = "Invalid entry";   //Reason for rejecting request
   ⋮
status = JNH_reject_revocation(id, reason);
return status;
```

⋮

The object in the RA’s object store that corresponds to the revocation request is now flagged as “Revreq rejecting” and is removed when the EE receives the message.

Chapter 6. Writing applications for the certificate authority

This chapter provides information for developing applications that run on the CA component. It includes sample code for the different CA tasks and discusses special considerations for CA application development.

Approving and rejecting certificate requests

The CA issues a certificate in response to a certificate request from an EE. The CA approves a certificate request with a call to JNH_create_certificate and rejects it with JNH_reject_registration.

After the CA approves a certificate request through a call to JNH_create_certificate, the certificate request information is passed to the RA and then to the EE. The surrogate object in the CA’s object store is flagged “Signed” and is removed from the object store when the RA receives the information.

The following code is an example of a CA approving a certificate request and creating a certificate:

```
uint32 id = 1;      //Request identifier
uint32 KeyID = 1;   //CA Key ID
uint32 status;
   ⋮

status = JNH_create_certificate(id, KeyID);
return status;
   ⋮
```

If a CA rejects a certificate request, the certificate request information and the reason for rejection are passed to the RA and then to the EE. The surrogate object in the CA’s object store is flagged “Rejected” and is removed from the object store when the RA receives the information.

The following code is an example of a CA rejecting a certificate request:

```
uint32 id = 1;                                //Request identifier
utf8String rejectReason[] = "Invalid data";   //Reason for the rejection
uint32 status;
   ⋮

status = JNH_reject_registration(id, rejectReason);
return status;
   ⋮
```

Revoking certificates

Certificates are revoked when their validity period is over or if the administrator suspects they have been compromised. After an RA approves and forwards a revocation request to the CA, the CA can either approve the request with JNH_revoke_certificate or reject it with JNH_reject_revocation. If a CA approves a revocation request and revokes the certificate, the certificate is added to a certificate revocation list (CRL).

Approving a revocation request

The CA approves a revocation request through a call to JNH_revoke_certificate, which revokes the certificate, adds the certificate to the CRL, and passes the information back to the RA. After this occurs, the surrogate object in the CA's object store is flagged "Revreq Accepted" and is removed from the object store when the RA receives the information.

The following example shows a CA approving a revocation request and revoking a certificate:

```
uint32 id = 1; //Request identifier
uint32 status;
   ⋮

status = JNH_revoke_certificate(id);
return status;
   ⋮
```

Rejecting a revocation request

A CA rejects a revocation request using JNH_reject_revocation, which passes to the RA the information from the request and a reason for its rejection. The surrogate object in the CA's object store is flagged "Revreq Rejected" and is removed from the object store when the RA receives the information.

The following code is an example of a CA rejecting a revocation request:

```
uint32 id = 1;                                //Request identifier
utf8String rejectReason[] = "Invalid data";   //Reason for the rejection
uint32 status;
   ⋮

status = JNH_reject_registration(id, rejectReason);
return status;
   ⋮
```

Creating a certificate revocation list (CRL)

Although JNH_revoke_certificate adds a certificate and its information to the CRL, no revocation is final until the CRL is issued. CRLs are published on a regular basis, determined by the security policies of the organization, and list the version number, issuing CA name, number of revoked certificates, a CRL number, and a list of the revoked certificates. The list of revoked certificates contains the serial number of each certificate and the time and date the certificate was revoked.

Using the sample test program, testcrl, the following example displays the contents of a CRL:

```
----- CRL Printout ------
*********************************************************************
*  .....  CRL successfully constructed ......
*********************************************************************
*
*       Version Number: 1 (v2)
*       Issuer Name:  /C=us/O=IBM/OU=Jonah
*       Number of Revoked Certs:  9

----------------------------------------------------------------------*
*       Serial Number:  1
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:25:45

----------------------------------------------------------------------*
Serial Number:  2
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:25:45

----------------------------------------------------------------------*
Serial Number:  3
```

```
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:25:45

----------------------------------------------------------------------*
Serial Number:  4
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:50:21

----------------------------------------------------------------------*
Serial Number:  5
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:48:56

----------------------------------------------------------------------*
Serial Number:  6
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:48:58

----------------------------------------------------------------------*
Serial Number:  7
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:49:3

----------------------------------------------------------------------*
Serial Number:  8
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:49:1

----------------------------------------------------------------------*
Serial Number:  9
*       CRT Revoked on (Y/M/D/H:Min:Sec) GMT: 1999/4/23/21:49:6
*
*********************************************************************
```

If a CA processes a revocation request prior to the next scheduled CRL, the CA can create a CRL through a call to JNH_create_CRL. The following code is an example of a CA creating a CRL:

```
uint32 status;
   ⋮

status = JNH_create_CRL();
return status;
   ⋮
```

Chapter 7. Application Programming Interfaces

The following section contains descriptions of the PKIX interfaces used to create and manage certificates.

Note: Many of the APIs are available in both C++ and Java interfaces. In the programming reference information, the syntax of both interfaces is shown, where applicable. In cases where only one interface syntax is shown, the API information is available for only the C++ interface.

The following APIs are used to register users and create certificates:

API Name	Purpose
JNH_authorize_registration	Authorizes a user registration request.
JNH_create_certificate	Creates a certificate for a user.
JNH_preregister_user	Prepares for a user registration.
JNH_publish_certificate	Publishes a certificate to an LDAP directory.
JNH_RA_preregister_user	Creates a preregistration record for an end-user registration.
JNH_register_user	Submits a user registration request.
JNH_reject_registration	Rejects a user registration request.
JNH_validate_registration	Validates that the version and signing algorithm are present in a certificate request.

The following APIs are used for revoking certificates and creating Certificate Revocation Lists (CRLs).

API Name	Function
JNH_authorize_revocation	Approves a revocation request for a certificate.
JNH_cancel_revreq	Cancels a revocation request.
JNH_create_CRL	Creates a certificate revocation list.
JNH_create_revreq	Creates a revocation request.
JNH_create_revreq_from_certificate	Creates a revocation request for a certificate on a smart card.
JNH_new_revreq	Creates a new revocation request.
JNH_publish_CRL	Publishes a certificate revocation list to the RA.
JNH_reject_revocation	Rejects a revocation request.
JNH_request_CRL	Requests a CRL.
JNH_request_revocation	Creates a revocation request.
JNH_revoke_certificate	Revokes a certificate.

The following APIs are used to obtain information about certificate requests in the object store:

API Name	Function
JNH_inquire_certreq_basicConstraints	Retrieves the basic constraints.

API Name	Function
JNH_inquire_certreq_enddate	Retrieves the expiration date.
JNH_inquire_certreq_issuer	Retrieves the issuer name.
JNH_inquire_certreq_KeyUsage	Retrieves the key-usage information.
JNH_inquire_certreq_privkey_EE	Returns the key length and algorithm of the EE-generated private key.
JNH_inquire_certreq_serialnumber	Retrieves the serial number.
JNH_inquire_certreq_startDate	Retrieves the starting date.
JNH_inquire_certreq_status	Retrieves the status of a certificate request.
JNH_inquire_certreq_subject	Retrieves the subject name of a certificate request.
JNH_inquire_certreq_subjectkey_algorithm	Retrieves the subject key algorithm.

The following APIs are used to obtain information about certificates in the EE's key store.

API Name	Function
JNH_cert_inquire_enddate	Retrieves the expiration date.
JNH_cert_inquire_issuer	Retrieves the issuer name.
JNH_cert_inquire_keyUsage	Retrieves the key usage information.
JNH_cert_inquire_serialNumber	Retrieves the certificate serial number.
JNH_cert_inquire_startDate	Retrieves the starting date.
JNH_cert_inquire_subject	Retrieves the subject name.
JNH_Keypair_Selected	Selects a key pair from a list of key pairs.
JNH_keystore_inquire_cert	Retrieves a certificate from the key store.
JNH_List_Existing_Keypairs	Lists any existing key pairs.
JNH_list_SC_certs	Returns the subject names and key identifiers for all the certificates in the key store.

The following APIs are used to modify certificate requests.

API Name	Function
JNH_add_certreq_extension	Adds an unsupported extension to a certificate request.
JNH_modify_certreq_extension	Modifies certificate extensions.
JNH_remove_certreq_extension	Removes an extension from a certificate extension.
JNH_set_certreq_basicConstraints	Sets the basic constraints.
JNH_set_certreq_endDate	Sets the expiration date.
JNH_set_certreq_issuer	Sets the issuer name.
JNH_set_certreq_keyUsage	Sets the key-usage.
JNH_set_certreq_privkey_EE	Sets the private key algorithm and key length of an EE.
JNH_set_certreq_startDate	Sets the starting date.
JNH_set_certreq_subject	Sets the subject name.

The following APIs are used to retrieve information about revocation requests.

API Name	Function
JNH_inquire_revreq_certIssuer	Returns the name of the certificate issuer.
JNH_inquire_revreq_certserialnumber	Returns one of the certificate serial numbers from the revocation request.
JNH_inquire_revreq_certserialnumbers	Returns a list of the certificate serial numbers for the certificates in a revocation request.
JNH_inquire_revreq_hold_instruction_code	Returns the hold_instruction_code from one of the certificates being revoked.
JNH_inquire_revreq_invalidityDate	Returns the invalidity date from one of the certificates being revoked.
JNH_inquire_revreq_reason	Returns the reason a certificate is being revoked.
JNH_inquire_revreq_requests	Determines how many certificates are being revoked in a revocation request.

The following APIs are used to modify revocation requests.

API Name	Function
JNH_set_revreq_certIssuer	Sets the name of the certificate issuer in the CertDetails section of the revocation request.
JNH_set_revreq_certserialnumber	Sets the certificate serial number of a certificate being revoked.
JNH_set_reverq_hold_instruction_code	Sets the hold_instruction_code for a certificate being revoked.
JNH_set_revreq_invalidityDate	Sets the invalidity date for a certificate being revoked.
JNH_set_revreq_reason	Sets the reason a certificate is being revoked.

The following APIs are used to create and manage browser-based certificate requests:

API Name	Function
JNH_CA_nonpkix_user_check	Checks the status of a certificate request submitted by JNH_CA_register_nonpkix_user.
JNH_CA_nonpkix_user_done	Removes from the object store the certificate request created by JNH_CA_preregister_nonpkix_user.
JNH_CA_nonpkix_user_get_cert	Retrieves the certificate for a non-PKIX end entity.
JNH_CA_preregister_nonpkix_user	Creates an initialization request for a non-PKIX end entity.
JNH_CA_register_nonpkix_user	Submits a certificate request for a non-PKIX end entity.
JNH_RA_nonpkix_create_revreq	Creates a revocation request object for a non-PKIX end entity.
JNH_RA_nonpkix_request_revocation	Requests a revocation for a certificate for a non-PKIX end entity.

API Name	Function
JNH_RA_nonpkix_user_check	Checks the status of a certificate request submitted by a non-PKIX end entity.
JNH_RA_nonpkix_user_done	Removes from the object store the certificate request created by JNH_RA_preregister_nonpkix_user.
JNH_RA_nonpkix_user_get_cert	Retrieves the certificate for a non-PKIX end entity.
JNH_RA_preregister_nonpkix_user	Creates an initialization request for a non-PKIX end entity.
JNH_RA_register_nonpkix_user	Submits a certificate request for a non-PKIX end entity.

The following APIs are used to work with .ini files.

API Name	Function
JNH_INI_deleteKey	Deletes a key and section from an .INI file.
JNH_INI_deleteSection	Deletes a section from an .INI file.
JNH_INI_initialize	Opens an .INI object for use.
JNH_INI_readKeys	Returns a list of the keys in an .INI file section.
JNH_INI_readSections	Returns a list of the sections in an .INI file.
JNH_INI_readString	Reads a string from the Jonah.INI file.
JNH_INI_writeFile	Writes an .INI file to the medium.
JNH_INI_writeString	Writes a string into the Jonah.INI file.

The following APIs are used to import and export private keys and public certificates, in the form of PKCS #12 files.

Note: When using the PKCS #12 APIs, you must save the PKCS #12 file as binary.

API Name	Function
JNH_pkcs12scExportFile	Exports a private key and public certificate from the smart card.
JNH_pkcs12UserExportFile	Exports a private key and public certificate from the object store.
JNH_pkcs12ImportFile	Imports a private key and public certificate to the smart card.

The following APIs are used for cross-certification of a CA.

API Name	Function
JNH_CA_list_signing_keys	Lists the subject CA's signing key.
JNH_preregister_crosscert	Creates a cross-certification record for a CA, using the preregistration record created by JNH_RA_preregister_crosscert.
JNH_RA_preregister_crosscert	Creates a preregistration record for a subject CA. This record is used for cross-certification with another CA.

API Name	Function
JNH_Set_CCert_VerificationKey	Sets the subject CA’s signing key for cross-certification.
JNH_subject_submit_crosscert	Submits a subject CA’s request for cross-certification.

The following APIs are used to perform various other tasks, including bootstrap and RA enrollment, related to objects and certificates.

API Name	Function
JNH_BootStrap	Performs a bootstrap on an RA or CA.
JNH_CA_list_RAs	Lists the RAs trusted by a CA.
JNH_CA_write_info	Creates a file in which to store a CA self-signed certificate.
JNH_confirm_msg	Send confirmation message to the specified object.
JNH_create_BootStrap	Creates a bootstrap request for an RA or CA.
JNH_create_enrollment_request	Creates an enrollment request.
JNH_delete_object	Deletes an object.
JNH_enroll	Sends an RA enrollment request to the CA.
JNH_enroll_RA	Sends enrollment request from a CA to an RA.
JNH_export_credential	Exports a certificate or key to a PKCS#12 file, virtual smart card, or a real smart card.
JNH_get_CA_info	Finds a CA’s URL, certificate subject name, and serial number.
JNH_get_error	Retrieves text associated with a message code.
JNH_get_fingerprint	Returns an RA or CA’s fingerprint.
JNH_get_object_state	Retrieves the status for an object.
JNH_get_self_serial_number	Returns an array of serial numbers needed for JNH_get_fingerprint.
JNH_get_self_subjectKeyInfo	Returns key information, public key, and algorithms from a self-signed certificate.
JNH_GetStatus	Returns the current status of a server.
JNH_initialize_UI	Sets up communication with a background server.
JNH_list_objects	Causes all active objects to announce themselves.
JNH_list_surrogates	Causes all surrogate objects to announce themselves.
JNH_register_callbacks	Registers the callbacks for user-written JNH_Notify and JNH_Display routines..
JNH_release_object	Unlocks an object in the object store without first saving it.
JNH_release_octetString	Releases the memory associated with an octet string.

API Name	Function
JNH_reserve_object	Locks an object in the object store.
JNH_save_object	Saves and unlocks an object in the object store.
JNH_server_login_pwd	Unlocks a server’s credentials by directly supplying a PIN.
JNH_set_RA_URL	Sets the RA URL, subject, and password in a revocation request.
JNH_set_server_location	Sets the background server location and checks that the server is running in that location.
JNH_shutdown_UI	Allows the server to clean up resources associated with a background server user interface session.
JNH_start_server	Initializes a server and opens or creates an object store.
JNH_stop_server	Shuts down a server in an orderly fashion.
JNH_store_RA_URL	Extracts the RA URL from the object store and stores it in an .ini file.

JNH_add_certreq_extension

Adds an extension to a certificate request.

Syntax

C++

```
uint32 JNH_add_certreq_extension(uint32 xtype,
                                 uint32 reqId,
                                 utf8String extId,
                                 octetString value,
                                 int critical)
```

Java

```
int JAVA_add_certreq_extension(int xtype,
                               int reqId,
                               String extId,
                               String value,
                               int critical)
```

Parameters

xtype – input
The type of extension to add:

1	HOST_ID_MAPPING
2	KEY_USAGE
3	SUBJECT_ALT_NAME
4	PRIVATE

reqId – input
The identifier of the certificate request.

extId – input
The identifier of the extension to add to the certificate request. This parameter is primarily used for private extensions, where it is provided in a dot string format (such as 1.4.5.12).

value – input
The octetString value of the extension.

The syntax for the certificate extension's value parameter follows:

HOST_ID_MAPPING
The value parameter should point to a character string. This string will contain a host name, a user name, and optionally a password, all separated by '/' characters.

KEY_USAGE
The value parameter should point to a character string which is nine bytes long. Each character should have the value 1 or 0, representing the setting of the corresponding bit in the actual extension. The offsets for each flag are as follows:

0	digitalSignature
1	nonRepudiation
2	keyEncipherment
3	dataEncipherment

4	keyAgreement
5	keyCertSign
6	cRLSign
7	encipherOnly
8	decipherOnly

EE programs should never set keyCertSign or cRLSign. The offsets match those given for bits in this extension in RFC 2459 and X.509.

SUBJECT_ALT_NAME
The value parameter should point to a character string. If the character string is a legal IP address, it is interpreted as an IP address. If the character string is a legal RFC-822 e-mail address (for example, name@domain), it is interpreted as one. If the character string is a legal internet domain name, it is interpreted as one. If the character string is a legal URI (including a legal URL), it is interpreted as one. If the character string is in OSF syntax for a DN, it is interpreted as a DN.

PRIVATE
The value parameter should be a DER-encoded binary object, whose ASN.1 definition matches the SYNTAX parameter of the extension definition. DER (Distinguished Encoding Rules) is one of the standard encodings used for ASN.1 objects, and it is defined in ITU standard X.690 (formerly X.209) and in ISO standard 8825. It differs from the more familiar BER (Basic Encoding Rules) in that constructed strings and indefinite lengths are not permitted.

critical – input
Determines whether or not the extension is critical:

0	no
1	yes

Note: Private extensions must not be critical.

Usage

RA, CA, EE

Remarks

Only one Key Usage and Subject Alternative Name extension identifier can be added to each certificate. A host id mapping extension can be added for each unique host name, and a private extension can be added for each unique extension identifier.

Return Values

0 Normal, successful completion

> 0
An error has occurred. See the apimsg.h file for details.

Related Functions

- JNH_modify_certreq_extension

Example

C++

```
octetString value;
int critical= 0
xtype = SUBJECT_ALT_NAME;
utf8String extId = "2.5.29.17";
value.data = (unsigned char *)strdup("127.0.0.1");
value.length = strlen((char *)value.data);
uint32 rc = JNH_add_certreq_extension(xtype, reqId, extId, value, critical);
JNH_release_octetString(value);  // Release octetString associated with value
```

Java

```
int retVal;
int xtype = 1;
int reqId;     // id of certificate request
String extId= "2.5.29.17";
String value= "127.0.0.1";
int critical = 0;
retVal = jonahInterface.JAVA_add_certreq_extension(xtype,
         reqId, extId, value, critical);
```

JNH_authorize_registration

Used by the RA to approve a certificate request and send it to the CA.

Syntax

C++

```
uint32 JNH_authorize_registration(uint32 reqId, uint32 keyId)
```

Java

```
int JAVA_Authorize_Registration(int reqId, int keyId)
```

Parameters

reqId – input
The identifier of the registration request

keyId – input
The identifier of the key.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_reject_registration
- JNH_register_user

Example

C++

```
uint32 ret;
uint32 status;
uint32 reqid;     // id of the certificate request
if(status == objstRACertReqActive)
   ret = JNH_authorize_registration(reqId, 0);
```

Java

```
int reqId;    //id of certificate request
int retVal = jonahInterface.JAVA_Authorize_Registration(reqId, 0);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error (retVal);
   System.out.println ("Authorize Registration Error= " +
                        jonahInterface.retStr);
}
```

JNH_authorize_revocation

Authorizes a revocation request for a certificate and sends the request on to the CA. This routine also sets the status of the revocation request.

Syntax

C++

```
uint32 JNH_authorize_revocation(uint32 reqId)
```

Java

```
int JAVA_authorize_revocation(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_request_revocation

Example

C++

```
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_authorize_revocation(reqId);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_authorize_revocation(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   system.out.println("Authorize_revocation error= " +
                       jonahInterface.retStr);
}
```

JNH_BootStrap

Performs a bootstrap on an RA or CA using an object created by the bootstrap.

Syntax

```
uint32 JNH_BootStrap(uint32 reqId,
                     const utf8String uPin)
```

Parameters

reqId – input
The identifier of the bootstrap request.

uPin – input
The user PIN specified to initsc for the RA or CA.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The following example shows a high-level flow of the sequence in which example JNH_Bootstrap is issued:

```
JNH_start_server(server_type)         // start up entity
JNH_GetStatus(&serverstatus)          // get status of entity
if (serverstatus == svrSt_BootStrap)     // if the entity requires Bootstrap,
   status = JNH_create_Bootstrap(&objid)  // then create the Bootstrap request
   if (status == OK)                      // if create Bootstrap request was successful,
     JNH_BootStrap               // then Bootstrap
```

Related Functions

- JNH_create_BootStrap

Example

```
uint32 status;
uint32 reqId;     // Bootstrap request identifier
status = JNH_BootStrap(reqId, (utf8String) "yourPin");
```

JNH_CA_list_RAs

Returns the names of the RAs currently trusted by the CA.

Syntax

```
uint32 JNH_CA_list_RAs(uint32 * numRAs,
                       utf8String ** raNames,
                       utf8String ** tags)
```

Parameters

numRAs – output
The number of RAs trusted by the CA.

raNames – output
The names of the RAs trusted by the CA.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

```
utf8String * raNames;
uint32 numRAs;
utf8String * tags
int rc;
rc = JNH_CA_list_RAs(&numRAs, &raNames, &tags);
```

JNH_CA_list_signing_keys

Lists the subject CA’s signing key for cross-certification.

Syntax

```
uint32 JNH_CA_list_signing_keys(uint32* numKeys,
                                utf8String** algoNames,
                                utf8String** keyLens,
                                utf8String**serialNums)
```

Parameters

numKeys – output
The number of signing keys.

algoNames – output
An array containing the algorithm name.

keyLens – output
An array containing the length of the signing key.

serialNums – output
An array containing the serial number of the signing key.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_CA_list_RAs

Example

```
uint32 numKey;
utf8String* algoName;
utf8String* serialNum;
utf8String* keyLen;
status = JNH_CA_list_signing_keys(&numKey, &algoName, &keyLen, &serialNum);
cout << "JNH_CA_list_signing_keys returns " << status << " and numKey
" << numKey << endl;

if (status == 0) {
   for (int i = 0; i < numKey; i++) {
      cout << "i = " << i << "\n";
      cout << "algoName = " << algoName[i] << "\n";
      cout << "keyLength = " << keyLen[i] << "\n";
      cout << "serialN = " << serialNum[i] << "\n" << endl;
   }
} else {
   return -1;
}
```

JNH_CA_nonpkix_user_check

Checks on the status of the certificate request submitted by JNH_CA_register_nonpkix_user.

Syntax

```
uint32 JNH_CA_nonpkix_user_check(uint32 reqId,
                                 octetString * RequestStatus)
```

Parameters

reqId – input
The identifier of the certificate request.

RequestStatus – output
The status of the registration request:

1	JNH_CertRequestSubmitted
2	JNH_CertRequestApproved
3	JNH_CertRequestRejected
4	JNH_CertRequestError

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details

Remarks

This API is primarily used in response to a request from the non-PKIX end entity for status and is used for browser-based certificates.

Related Functions

- JNH_CA_register_nonpkix_user

Example

```
uint32 reqId;              // request identifier
uint32 RequestStatus       // request status
uint32 status;
status = JNH_CA_nonpkix_user_check(reqId, &RequestStat);
```

JNH_CA_nonpkix_user_done

Removes from the object store the certificate request created by JNH_CA_preregister_nonpkix_user.

Syntax

```
uint32 JNH_CA_nonpkix_user_done(uint32 reqId)
```

Parameters

reqId – input
The identifier of the certificate request to be removed from the object store.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_CA_preregister_nonpkix_user

Example

```
uint32 reqId;         // request identifier
uint32 status;
status = JNH_CA_nonpkix_user_done(reqId);
```

JNH_CA_nonpkix_user_get_cert

Retrieves the certificate for a non-PKIX end entity.

Syntax

```
uint32 JNH_CA_nonpkix_user_get_cert(uint32 reqId,
                                    octetString ** Certificate)
```

Parameters

reqId – input
The identifier of the certificate request created by JNH_CA_preregister_nonpkix_user.

Certificate – output
A pointer to where the certificate (as an octetString) is stored.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_CA_preregister_nonpkix_user

Example

```
uint32 reqId;              // request identifier
octetString Certificate;   // pointer to where the certificate is stored
uint32 status;
status = JNH_CA_nonpkix_user_get_cert(reqId, &Certificate);
```

JNH_CA_preregister_nonpkix_user

Creates a certificate request for a non-PKIX end entity.

Syntax

```
uint32 JNH_CA_preregister_nonpkix_user(const utf8String UserName,
                                       uint32 * reqId)
```

Parameters

UserName – input
The Distinguished Name of the non-PKIX end entity. UserName is a Distinguished Name in OSF syntax as described in "Parameter format" on page 7.

reqId – output
A pointer to where the identifier of the new certificate request is written.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_CA_register_nonpkix_user

Example

```
uint32 reqId;
uint32 status;
status = JNH_CA_preregister_nonpkix_user("/C=US:, "CN=A_USER", 0)
```

JNH_CA_register_nonpkix_user

Submits a certificate request for a non-PKIX end entity.

Syntax

```
uint32 JNH_CA_register_nonpkix_user(const uint32 reqId)
```

Parameters

reqId – input
The identifier of the certificate request created with JNH_CA_preregister_nonpkix_user.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_CA_preregister_nonpkix_user

Example

```
uint32 status;
uint32 reqId;     // request identifier
status = JNH_CA_register_nonpkix_user(reqId);
```

JNH_CA_write_info

Creates a file in which to store a CA self-signed certificate.

Syntax

C++

```
uint32 JNH_CA_write_info(const utf8String filename)
```

Java

```
int JAVA_CA_write_info (String filename)
```

Parameters

filename – input
The name of the file to create.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_enroll

Example

C++

```
uint32 status;
const utf8String outfile = "out.file";

status = JNH_CA_write_info(outfile);
```

Java

```
String outfile = "out.file";
int retval;
retval = jonah.Interface.JAVA_CA_write_info (outfile);
```

JNH_cancel_revreq

Cancels a revocation request.

Syntax

C++

```
uint32 JNH_cancel_revreq(uint32 reqId)
```

Java

```
int JAVA_cancel_revreq(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request to cancel.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_revreq

Example

C++

```
uint32 status;
status = JNH_cancel_revreq(uint32 reqId);
```

Java

```
int retVal;
retVal = jonahInterface.JAVA_cancel_revreq(element.elemId);
```

JNH_cert_inquire_endDate

Returns the ending date for the supplied certificate.

Syntax

C++

```
uint32 JNH_cert_inquire_endDate(const octetString cert_buffer,
                                utcDateTime * endDate);
```

Java

```
int JAVA_cert_inquire_endDate(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate to be queried.

endDate – output
The ending date of the certificate.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_cert_inquire_startdate

Example

C++

```
uint32 retVal;
utcDateTime  end_date;
octetString cert_buffer;

retVal = JNH_cert_inquire_endDate(cert_buffer, &endDate );
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                             keyIDVector,
                                                             pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_endDate(certBuffer);

if (retVal == 0)
{

   year = frameMain.jonahInterface.year;
   month = frameMain.jonahInterface.month;
   day = frameMain.jonahInterface.day;

   endDate = month + "/" + day + "/" + year;
}
```

JNH_cert_inquire_issuer

Returns the certificate issuer for the supplied certificate.

Syntax

C++

```
uint32 JNH_cert_inquire_issuer(const octetString cert_buffer,
                               utf8String * issuer);
```

Java

```
int JAVA_cert_inquire_issuer(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate being queried.

issuer – output
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_cert_inquire_subject

Example

C++

```
uint32 retVal;
utf8String issuer;
octetString cert_buffer;

retVal = JNH_cert_inquire_issuer(cert_buffer, &issuer );
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                             keyIDVector,
                                                             pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_issuer(certBuffer);

if (retVal == 0)
    //Issuer Name.
    issuer = frameMain.jonahInterface.retStr;
```

JNH_cert_inquire_keyUsage

Returns the key usage extension information for the supplied certificate.

Syntax

C++

```
uint32 JNH_cert_inquire_keyUsage(const octetString cert_buffer,
                                 keyUsage_t * usages);
```

Java

```
int JAVA_cert_inquire_keyUsage(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate to be queried.

usages – output
The key usage extension information:

Extension	**Usage**
1	Digital Signature
2	Non-repudiation
4	Key Encipherment
8	Data Encipherment
16	Key Agreement
32	Key Cert Sign
64	CRL Sign
128	Encipher Only
256	Decipher Only

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_keystore_inquire_cert

Example

C++

```
uint32 retVal;
keyUsage_t usages;
octetString cert_buffer;

retVal = JNH_cert_inquire_keyUsage(cert_buffer, &usages);
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                             keyIDVector,
                                                             pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_keyUsage(certBuffer);

if (retVal == 0)
   keyUsage = frameMain.jonahInterface.retInt;
```

JNH_cert_inquire_serialNumber

Returns the serial number for the supplied certificate.

Syntax

C++

```
uint32 JNH_cert_inquire_serialNumber(const octetString cert_buffer,
                                     octetString * serialNumber);
```

Java

```
int JAVA_cert_inquire_serialNumber(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate to be queried.

serialNumber – output
The serial number of the certificate, a long integer stored in the octetString data buffer. This buffer needs to be cast to long before it can be used.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_cert_inquire_subject

Example

C++

```
uint32 retVal;
octetString serial_number;
octetString cert_buffer;

retVal = JNH_cert_inquire_serialNumber(cert_buffer, &serial_number );
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                            keyIDVector,
                                                            pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_serialNumber(certBuffer);

if (retVal == 0)
   serialNumber = frameMain.jonahInterface.retStr;
```

JNH_cert_inquire_startDate

Returns the starting date for the supplied certificate.

Syntax

C++

```
uint32 JNH_cert_inquire_startDate(const octetString cert_buffer,
                                  utcDateTime * start_date);
```

Java

```
int JAVA_cert_inquire_startDate(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate being queried.

start_date – output
The starting date of the certificate.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_cert_inquire_endDate

Example

C++

```
uint32 retVal;
utcDateTime  start_date;
octetString cert_buffer;

retVal = JNH_cert_inquire_startDate(cert_buffer, &start_date );
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
              keyIDVector,
              pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_startDate(certBuffer);

if (retVal == 0)
{

   year = frameMain.jonahInterface.year;
   month = frameMain.jonahInterface.month;
   day = frameMain.jonahInterface.day;

   startDate = month + "/" + day + "/" + year;
}
```

JNH_cert_inquire_subject

Parses the subject name of a DER-encoded certificate buffer.

Syntax

C++

```
uint32 JNH_cert_inquire_subject(const octetString cert_buffer,
                                utf8String * subject);
```

Java

```
int JAVA_cert_inquire_subject(byte[] cert_buffer)
```

Parameters

cert_buffer – input
The buffer containing the certificate being queried.

subject – output
The subject of the certificate, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_keystore_inquire_cert

Example

C++

```
uint32 retVal;
utf8String subject;
octetString cert_buffer;

retVal = JNH_cert_inquire_subject(cert_buffer, &subject );
```

Java

```
retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                             keyIDVector,
                                                             pin);

byte[] certBuffer;

certBuffer = frameMain.jonahInterface.retByteArr;

retVal = frameMain.jonahInterface.JAVA_cert_inquire_subject(certBuffer);

if (retVal == 0)
    // String object that contains the certificates subject name.
    subject = frameMain.jonahInterface.retStr;
```

JNH_cleanup_BootStrap

Cleans up the information allocated for bootstrap. This call is used to abort the bootstrap operation.

Syntax

```
uint32 JNH_cleanup_BootStrap(uint32* reqId)
```

Parameters

reqId – input
The identifier of the bootstrap request. If you’ve not created a bootstrap request yet, pass in NULL.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_bootstrap
- JNH_BootStrap

Example

```
status = JNH_cleanup_BootStrap(&reqId)
```

JNH_confirm_msg

Send a confirmation message for the specified object.

Syntax

C++

```
uint32 JNH_confirm_msg (uint32 reqId)
```

Java

```
int JAVA_confirm_msg (int reqId)
```

Parameters

reqId – input
The identifier of the object for which a confirmation message is being sent.

Usage

EE, RA, CA

Return Values

0 Normal, successful completion

>0 An error occurred. See the apimsg.h file for details.

Example

C++

```
uint32 status;
uint32 reqId
status = JNH_confirm_msg (reqId);
```

Java

```
int status;
int reqId;
status = JAVA_confirm_msg (reqId);
```

JNH_create_BootStrap

Creates a bootstrap request for an RA or CA.

Syntax

```
uint32 JNH_create_BootStrap(uint32 * reqId)
```

Parameters

reqId – output
The identifier of the bootstrap request.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_BootStrap

Example

```
uint32 status;
uint32 reqId;

status = JNH_create_BootStrap(&reqId);
```

JNH_create_certificate

Creates a certificate using the data in the specified request. The certificate will be signed and returned to the RA.

Syntax

C++

```
uint32 JNH_create_certificate(uint32 reqId,
                              uint32 CAkeyId)
```

Java

```
int JAVA_Create_Certificate(int reqId,
                            int KeyId)
```

Parameters

reqId – input
The identifier of the request.

CAkeyId – input
The identifier of the CA key to be used to sign the certificate.

Note: This parameter is not currently used.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_register_user

Example

C++

```
uint32 status;
uint32 reqId;     // id of the certificate request
if (status== ObjStCACertReqActive)
{
   JNH_create_certificate(reqId,0);
}
```

Java

```
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_Create_Certificate (reqId, 0);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error (retVal);
   System.out.println("Create Certificate error= " +
                      jonahInterface.retStr);
}
```

JNH_create_CRL

Creates a certificate revocation list.

Syntax

C++

```
uint32 JNH_create_CRL()
```

Java

```
int JAVA_create_CRL()
```

Parameters

None

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

Any revocation request objects that are bound into the CRL will be completed. The CRL will be sent to the RA for publication.

Related Functions

- JNH_authorize_revocation

Example

C++

```
uint32 rc;
rc = JNH_create_CRL();
```

Java

```
int retVal = jonahInterface.JAVA_create_CRL();
if (retVal != 0)
{
System.out.println("Create CRL error= " + jonahInterface.retStr);
```

JNH_create_enrollment_request

Checks the fingerprint against the certificate in either the file or the LDAP entry, then creates an enrollment request.

Syntax

```
uint32 JNH_create_enrollment_request(utf8String nameOrFile,
                                     IBOOL isFile,
                                     utf8String caFingerprint,
                                     uint32 * reqId)
```

Parameters

nameOrFile – input
The location (either a file or an LDAP entry) where the certificate is stored.

isFile – input
Specifies whether the certificate location is a file or LDAP entry:

true	File
false	LDAP entry

caFingerprint – input
A string of characters known to the CA runtime that can be queried by the CA GUI and communicated to the user. This parameter is used to verify that a CA is legitimate.

reqId – output
The identifier of the enrollment request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_enroll

Example

```
uint32 reqId;
buffer_t buff;
uint32 status;
status = JNH_create_enrollment_request(
         (unsigned char *) "c:\\ca.info", true,
         buff.data, &reqId);
```

JNH_create_revreq

Creates a certificate revocation request.

Syntax

C++

```
uint32 JNH_create_revreq(uint32 reqId)
```

Java

```
int JAVA_create_revreq(int reqId)
```

Parameters

reqId – input
The identifier of the existing message, most often a certificate response, containing the certificate to be revoked.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_request_revocation

Example

C++

```
uint32 reqId;     // id of certificate request
JNH_create_revreq(reqId);
```

Java

```
int retVal;
int reqId;     // id of certificate request
retVal = jonahInterface.JAVA_create_revreq(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println("Error in create revreq= "
                       + jonahInterface.retStr);
}
```

JNH_create_revreq_from_certificate

Creates a revocation request for the given certificate.

Syntax

C++

```
uint32 JNH_create_revreq_from_certificate(const octetString certBuffer,
                                          uint32 * reqId)
```

Java

```
int JAVA_create_revreq_from_certificate(byte[] cert_buffer)
```

Parameters

certBuffer – input
An octet string containing the DER encoding of the certificate to be revoked.

reqId – output
The identifier of the new revocation request.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This call creates a revocation request in the EE's object store. The revocation request accessor functions can then be used to modify parameters of the request and JNH_request_revocation can then be used to submit the revocation request to the RA.

Related Functions

- JNH_create_revreq
- JNH_request_revocation

Example

C++

```
uint32 status;
octetString certBuffer;
uint32 reqID;
status = JNH_create_revreq_from_certificate(certBuffer, &reqID)
```

Java

```
int retVal = -1;
byte[] cert_buffer;
int reqID;

retVal = JAVA_create_revreq_from_certificate(cert_buffer);
```

JNH_delete_object

Deletes an object from the object store.

Syntax

C++

```
uint32 JNH_delete_object(uint32 reqId)
```

Java

```
int JAVA_delete_object(int reqId)
```

Parameters

reqId – input
The identifier of the object to be deleted.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

For a certificate or revocation request, this should only be done by the original requestor once the certificate has been issued or the revocation accepted. Deleting the request earlier will prevent the operation from completing.

Related Functions

- JNH_save_object

Example

C++

```
uint32 rc;
uint32 reqId;     // id of object
rc = JNH_delete_object(reqId);
```

Java

```
int retVal;
int reqId;     // id of request object
retVal = jonahInterface.JAVA_delete_object(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println("delete object error= " + jonahInterface.retStr);
}
```

JNH_enroll

Used to send the RA enrollment request to the CA.

Syntax

C++

```
uint32 JNH_enroll(uint32 objId)
```

Java

```
int JAVA_enroll (int objId)
```

Parameters

objId – input
The identifier of the RA enrollment request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_enrollment_request
- JNH_enroll_RA

Example

C++

```
uint32 status;
uint32 objId;
status = JNH_enroll(objId);
```

Java

```
int status;
int objId
status = JAVA_enroll(objId);
```

JNH_enroll_RA

Used by the CA to enroll an RA.

Syntax

C++

```
uint32 JNH_enroll_RA(uint32 reqId,
                     const utf8tring raURL, const utf8String raFingerprint)
```

Java

```
int JAVA_enroll_RA (int reqId, String raURL String raFingerprint)
```

Parameters

reqId – input
The identifier of the object to receive the message.

raURL – input
The RA’s URL.

raFingerprint – input
The RA’s fingerprint information.

Usage

CA

Return Values

0 Normal, successful completion

>0 An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_enroll

Example

C++

```
utf8String rafgPrint;
status = JNH_enroll_RA(objId, (const utf8String) x://localhost:829,
                              (utf8String) rafgPrint);
```

Java

```
int status;
status = JAVA_enroll_RA (reqId, raURL, raFingerprint);
```

JNH_export_credential

Exports the certificate or the private key to either a PKCS#12 file, virtual smart card, or real smart card (for end entity only).

Syntax

C++

```
uint32 JNH_export_credential(uint32 reqId,
                             const utf8String password,
                             const utf8String device,
                             IBOOL reuse_key=0)
```

Java

```
int JAVA_export_credential(int reqId,
                           String password,
                           String device)
```

Parameters

reqId – input
The identifier of the request.

password – input
The password to be associated with the credential.

device – input
Specifies the device to which the certificate is being exported. Must be one of the following:

VSC[:fileName]	virtual smart card
TOK[:readerName]	real smart card
PKCS12:fileName	PKCS#12 file

reuse_key – input
When true, the existing key pair from the smart card is to be reused. The default is set to false.

Usage

EE, RA, CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

C++

```
uint32 reqId;     // id of request object
uint32 rc;
rc = JNH_export_credential(reqId,
                           (utf8String) "foo",
                           (utf8String) "VSC:c://token.fil");
```

Java

```
String device= "PKCS12:c:\\PKCS12.OUT";
String password= "foo";
int retVal;
int reqId;     // id of request object
retVal = jonahInterface.JAVA_export_credential(reqId, password, device);
```

JNH_get_CA_info

Using the Distinguished Name or CA file name, finds the CA URL and the subject name and serial number for a certificate.

Syntax

```
uint32 JNH_get_CA_info(const utf8String nameOrFile,
                       IBOOL isFile,
                       utf8String * caURL,
                       utf8String * subjectName,
                       octetString * serialNumber)
```

Parameters

nameOrFile – input
The location (either a file or an LDAP entry) where the certificate is stored.

isFile – input
Specifies whether the certificate location is a file or LDAP entry:

true	File
false	LDAP entry

caURL – output
The URL of the CA.

subjectName – output
The subject name in a CA's certificate. subjectName is a Distinguished Name in OSF syntax as described in "Parameter format" on page 7.

serialNumber – output
The serial number in a CA's certificate.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

```
utf8String caURL, subjectName;
octetString serialNumber;
uint32 status;
status = JNH_get_CA_info((unsigned char * ) "c:\\ca.info,
         true, &caURL, &subjectName, &serialNumber);
```

JNH_get_error

Retrieves text associated with an error code (returned by a JNH_*xxx*

routine).

Syntax

C++

```
uint32 JNH_get_error(uint32 errorCode,
                     utf8String * errorText)
```

Java

```
int JAVA_get_error(int msgId)
```

Parameters

errorCode – input
Specifies the error code for which the text is being returned.

errorText – output
The text associated with the error code.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_register_callbacks

Example

C++

```
utf8String errorText=NULL;
uint32 status = API_INVALID_ARGUMENT;
uint32 rc;
rc = JNH_get_error (status, &errorText);
```

Java

```
int retVal;
retVal = jonahInterface.JAVA_create_CRL():
if (retVal != 0)
{
   jonahInterface.JAVA_get)_error(retval);
   System.out.println("Create CRL error= " + jonahInterface.retStr);
}
```

JNH_get_fingerprint

Returns an RA or CA’s fingerprint.

Syntax

```
uint32 JNH_get_fingerprint(const octetString serialNumber,
                           utf8String * fingerprint)
```

Parameters

serialNumber – input
The serial number returned by JNH_get_self_serial_numbers.

fingerprint – output
The fingerprint of the CA or RA.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_enrollment_request
- JNH_get_self_serial_numbers

Example

```
uint32 status;
octetString serialNum;
uff8String fingerprint;
status = JNH_get_fingerprint(*serialNum, &fingerprint);
```

JNH_get_IniMyName

Retrieves the value for the MyName field in the .ini file’s General section.

Syntax

```
uint32 JNH_get_IniMyName(utf8String* name)
```

Parameters

name – output
The MyName value.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_MyName

Example

```
utf8String myName;
status = JNH_get_IniMyName (&myName);
```

JNH_get_IniLdapAuthName

Retrieves the LDAP AuthName information from the .ini file.

Syntax

```
uint32 JNH_get_IniLdapAuthName(utf8String* authname)
```

Parameters

authname – output
The LDAP AuthName.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_IniLdapAuthName
- JNH_get_IniLdapServer

Example

```
utf8String ldapserveran;
status = JNH_get_IniLdapAuthName(&ldapserveran);
```

JNH_get_IniLdapAuthPwd

Retrieves the LDAP server password information from the .ini file.

Syntax

```
uint32 JNH_get_IniLdapAuthPwd(utf8String* pwd)
```

Parameters

pwd – output
The LDAP server password.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_IniLdapAuthPwd
- JNH_get_IniLdapAuthName

Example

```
utf8String passwd;
status = JNH_get_IniLdapAuthPwd(&passwd);
```

JNH_get_IniLdapServer

Retrieves the LDAP server information from the .ini file.

Syntax

```
uint32 JNH_get_IniLdapServer(utf8String* server)
```

Parameters

server – output
The LDAP server information.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_IniLdapServer
- JNH_get_IniTcpPort

Example

```
utf8String ldapserver;
status = JNH_get_IniLdapServer(&ldapserver);
```

JNH_get_IniTcpHost

Retrieves the TCP host information from the .ini file.

Syntax

```
uijnt32 JNH_get_IniTcpHost(utf8String* host)
```

Parameters

host – output
The TCP host name from the .ini file.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_IniTcpHost
- JNH_get_IniMyName

Example

```
utf8String tcphost;
status = JNH_get_IniTcpHost(&tcphost);
```

JNH_get_IniTcpPort

Retrieves the TCP port information from the .ini file.

Syntax

```
uint32 JNH_get_IniTcpPort(utf8String* port)
```

Parameters

port – output
The TCP port information.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_IniTcpPort
- JNH_get_IniTcpHost

Example

```
utf8String tcpport;
status = JNH_get_IniTcpPort(&tcpport);
```

JNH_get_self_serial_number

Returns an array of serial numbers needed to call JNH_get_fingerprint.

Syntax

```
uint32 JNH_get_self_serial_number(uint32 *monuments,
                                  octetString **serialNumbers)
```

Parameters

numElements – output
The number of serial numbers returned in the array.

serialNumbers – output
The array of serial numbers. The first serial number in the array is the self-signed serial number.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_get_fingerprint

Example

```
uint32 status;
uint32 numElements;
octetString * serialNum;
status = JNH_get_self_serial_numbers(&numElements, &serialNum);
```

JNH_get_self_subjectKeyInfo

Returns key information, public key, and algorithms from a self-signed certificate.

Syntax

```
uint32 JNH_get_self_subjectKeyInfo(uint32 *numElements,
                                   octetString **serialNumbers,
                                   utf8String **keyAlgorithms,
                                   uint32 **keyLengths)
```

Parameters

numElements – output
The number of elements returned.

serialNumber – output
An array of serial numbers, the first of which is the serial number of the self-signed certificate.

keyAlgorithms – output
An array of key algorithms.

keyLengths – output
An array of key lengths.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_subjectKeyInfo

Example

```
uint32 status;
uint32 numElements;
octetString * serialNum;
utf8String *keyAlgorithms = NULL;
uint32 *keyLenghts = 0;

status = JNH_get_self_subjectKeyInfo(&numElements, &serialNum,
         &keyAlgorithms, &keyLengths);
```

JNH_get_object_state

Returns the state of a specified object.

Syntax

```
uint32 JNH_get_object_state(uint32 objId,
                            uint32 * state)
```

Parameters

objId – input
The identifier of the object.

state – output
The state of the object.

Usage

EE, RA, CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The specified object must be locked with JNH_reserve_object before JNH_get_object_state can be used.

Related Functions

- JNH_register_callbacks

Example

```
uint32 objId;     //id of the object
uint32 state
uint32 rc;
rc = JNH_get_object_state(objId, &state);
```

JNH_GetStatus

Returns the current status of a server.

Syntax

```
uint32 JNH_GetStatus(uint32 * status)
```

Parameters

status – output
The status of the server:

svrSt_Recovery
Recovery

svrSt_UnEnroll
Unenrolled

svrSt_Bootstrap
Bootstrap

svrSt_Running
Running

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_get_object_state

Example

```
uint32 status;
uint32 serverstatus;

status = JNH_GetStatus(&serverstatus);
```

JNH_INI_deleteKey

Deletes the entry with the specified key and section from an .ini object.

Syntax

C++

```
uint32 JNH_INI_deleteKey(const utf8String section,
                         const utf8String key)
```

Java

```
int JAVA_INI_deleteKey(String section,
                       String key)
```

Parameters

section – input
The name of the section for the defined key.

key – input
The name of the key to be deleted.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_deleteSection

Example

C++

```
uint32 retVal;
utf8String section = "OIDs";
utf8String key = "C";

retVal = JNH_INI_deleteKey(section, key);
```

Java

```
int retVal = 0;
String section = "OIDs";
String key = "C";

retVal = jonahInterface.JAVA_INI_deleteKey(section, key);
```

JNH_INI_deleteSection

Deletes a specified section from an .ini object.

Syntax

C++

```
uint32 JNH_INI_deleteSection(const utf8String sectionName)
```

Java

```
int JAVA_INI_deleteSection(String sectionName)
```

Parameters

sectionName – input
The name of the section to be deleted.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_deleteKey

Example

C++

```
uint32 retVal;
utf8String section = "OIDs";

retVal = JNH_INI_deleteSection(section);
```

Java

```
int retVal = 0;
String section = "OIDs";

retVal = jonahInterface.JAVA_INI_deleteSection(section);
```

JNH_INI_Initialize

Opens an .ini object for use.

Syntax

C++

```
uint32 JNH_INI_Initialize(const utf8String iniName)
```

Java

```
int JAVA_INI_Initialize(String iniName)
```

Parameters

iniName – input
The name of the .ini object to open.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_readSection
- JNH_INI_deleteSection

Example

C++

```
uint32 retVal;
retVal = JNH_INI_Initialize("c:\\pkix\\templ\\caserver.ini");
```

Java

```
int retVal = jonahInterface.JAVA_INI_Initialize("c:\\pkix\\templ\\caserver.ini");
```

JNH_INI_readKeys

Returns a list of the keys found in a specified .ini object section.

Syntax

C++

```
uint32 JNH_INI_readKeys(const utf8String section,
                        utf8String ** plist,
                        uint32 * numKeys)
```

Java

```
int JAVA_INI_readKeys(String section)
```

Parameters

section – input
The name of the .ini section from which to retrieve the keys.

plist –
A pointer to a utf8String array to return the keys.

numKeys – output
The number of keys found in the specified section.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_writeFile
- JNH_INI_readSections
- JNH_INI_readString

Example

C++

```
utf8String * str;
uint32 numKeys;
uint32 retValInt;
utf8String section = "OIDs";

retValInt = JNH_INI_readKeys(section, &str, &numSections);
```

Java

```
String section = "OIDs";
int retVal = 0;

retVal = jonahInterface.JAVA_INI_readKeys(section);
if (retVal == 0)
{
   numKeys = jonahInterface.retStrArr.length;
   String iniKeys[] = new String[numKeys];
   System.arraycopy(jonahInterface.retStrArr, 0, iniKeys, 0, numKeys);
}
```

JNH_INI_readSections

Returns a list of the sections found in the .ini object.

Syntax

C++

```
uint32 JNH_INI_readSections(utf8String ** plist,
                            uint32 * numSections)
```

Java

```
int JAVA_INI_readSections()
```

Parameters

plist –input
A pointer to a utf8String array to return the section names.

numSections – output
The number of sections found in the .ini file.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_readString
- JNH_INI_readKeys

Example

C++

```
utf8String * str;
uint32 numSections;
uint32 retValInt;

retValInt = JNH_INI_readSections(&str, &numSections);
```

Java

```
int retVal = jonahInterface.JAVA_INI_readSections();
if (retVal == 0)
{
   int numSections = jonahInterface.retStrArr.length;
   String iniSections[] = new String[numSections];

   System.arraycopy(jonahInterface.retStrArr, 0, iniSections, 0, numSections);
}
```

JNH_INI_readString

Reads a string from the Jonah.ini file.

Syntax

C++

```
uint32 JNH_INI_readString(const utf8String section,
                          const utf8String key,
                          utf8String * value,
                          const utf8String defaultValue)
```

Java

```
int JAVA_INI_readString(String section,
                        String key)
```

Parameters

section – input
The section of the file from which to read the string.

key – input
The key to read in the specified section.

value – output
The value associated with the key.

defaultValue – input
The value to use if the key is not found.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_writeString
- JNH_INI_readKeys
- JNH_INI_readSections

Example

C++

```
utf8String value;
int rc;
rc = JNH_INI_readString(
        "Object Store",
        "Name",
        &value,
        "jonahee");
```

Java

```
int retVal = jonahInterface.JAVA_INI_readString("General", "TempPath");
if (retVal != 0)
{
   String TempPath = jonahInterface.retSt);
}
```

JNH_INI_writeFile

Writes an .ini object to the medium.

Syntax

C++

```
uint32 JNH_INI_writeFile()
```

Java

```
int JAVA_INI_writeFile()
```

Parameters

None

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_writeString

Example

C++

```
uint32 retVal;
retVal = JNH_INI_writeFile();
```

Java

```
int retVal = 0;
retVal = jonahInterface.JAVA_INI_writeFile();
```

JNH_INI_writeString

Writes the string to the .ini object.

Syntax

C++

```
uint32 JNH_INI_writeString(const utf8String section,
                           const utf8String key,
                           const utf8String value)
```

Java

```
int JAVA_INI_writeString(String section,
                         String key,
                         String value)
```

Parameters

section – input
The section of the file to write the string in.

key – input
The key to be added or updated in the defined section of the .ini object.

value – input
The value string assigned to the key.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_INI_readString
- JNH_INI_writeFile

Example

C++

```
uint32 rc;
rc = JNH_INI_writeString
           ("Object Store",
            "Name",
            "jonahee");
```

Java

```
int retVal = 0;
String section = "OIDs";
String key = "C";
String value = "1.2.3.4";

retVal = jonahInterface.JAVA_INI_writeString(section, key, value);
```

JNH_initialize_UI

Sets up communication with a background server.

Syntax

```
uint32 JNH_initialize_UI(octetString *certBuff,
                         const char * userPin)
```

Parameters

certBuff – input
The DER encoding of the certificate associated with an administrator.

userPin – input
The password for the local keystore. This parameter allows the smart card (virtual or real) to be used to sign messages to the server.

Usage

RA, CA

Return Values

REMJNH_OK
Normal, successful completion

REMJNH_NOCONNECTION
Unable to connect to a server at the given address and port.

Related Functions

- JNH_shutdown_UI

Example

```
octetString certBuff;
utf8String scPIN;

status = JNH_initialize_UI(&certBuff, scPIN);

if (status != 0) {
   //handle error conditions
}
```

JNH_inquire_certreq_basicConstraints

Retrieves the basic constraints from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_basicConstraints(uint32 reqId,
                                            IBOOL * isCaCert,
                                            uint32 * maxPathLen,
                                            IBOOL * isMaxPathLenUnlimited)
```

Java

```
int JAVA_inquire_certreq_basicConstraints(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

isCaCert – output
Determines if the certificate request is from a CA.

maxPathLen – output
Returns the maximum path length allowed. The default length is 0. A maxPathLen value of –1 means that the path length is unconstrained.

isMaxPathLenUnlimited – output
Specifies whether or not the maximum path length is unlimited.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_basicConstraints

Example

C++

```
uint32 reqId;     // id of certificate request
uint32 rc;
bool isCaCert;
uint32 maxPathLen
IBOOL isMaxPathLenUnlimited;
rc = JNH_inquire_certreq_basicConstraints(reqId, &isCaCert,
                          &maxPathLen, &isMaxPathLenUnlimited);
```

Java

```
int retVal;
int reqId;     // id of certificate request
retVal = jonahInterface.JAVA_inquire_certreq_basicContraints(reqId);
if (retVal != 0)
```

```
{
   int maxPathLen = jonahInterface.retInt;
   boolean isCaCert = jonahInterface.retBool;
}
```

JNH_inquire_certreq_enddate

Retrieves the expiration date from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_enddate(uint32 reqId,
                                   utcDateTime * enddate)
```

Java

```
int JAVA_inquire_certreq_enddate(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

enddate – output
The expiration date of the certificate request.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_startdate
- JNH_set_certreq_endDate

Example

C++

```
utcDateTime enddate;
uint32 reqId;     // id of certificate request
rc = JNH_inquire_certreq_enddate(reqId, &enddate);
```

Java

```
int reqId;     // id of certificate request
Date enddate;
int retVal = jonahInterface.JAVA_inquire_certreq_enddate(reqId);
if (retVal ==0)
{
   enddate.year = jonahInterface.year;
   enddate.month = jonahInterface.month;
   enddate.day = jonahInterface.day;
}
```

JNH_inquire_certreq_issuer

Retrieves the issuer name from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_issuer(uint32 reqId,
                                  utf8String * issuer)
```

Java

```
int JAVA_inquire_certreq_issuer(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

issuer – output
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_issuer

Example

C++

```
uint32 reqId;     // id of certificate request
utf8String issuer;
rc = JNH_inquire_certreq_issuer(reqId, &issuer);
```

Java

```
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_inquire_certreq_issuer(reqId);
if (retVal == 0)
{
   String issuerName = jonahInterface.retStr;
}
```

JNH_inquire_certreq_keyUsage

Retrieves the key usage extension information from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_keyUsage(uint32 reqId,
                                    keyusage_t * usages)
```

Java

```
int JAVA_inquire_certreq_keyUsage(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

usages – output
The key usage extension information:

Extension	**Usage**
1	Digital Signature
2	Non-repudiation
4	Key Encipherment
8	Data Encipherment
16	Key Agreement
32	Key Cert Sign
64	CRL Sign
128	Encipher Only
256	Decipher Only

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_keyUsage

Example

C++

```
uint32 rc;
uint32 reqId;     // id of certificate request
keyUsage_t usages;
rc = JNH_inquire_certreq_keyUsage(reqId, &usages);
```

Java

```
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_inquire_certreq_keyUsage(reqId);
if (retVal == 0)
{
   int KeyUsage = jonahInterface.retInt;
}
```

JNH_inquire_certreq_privkey_EE

Returns the key length and algorithm of the EE-generated private key for the specified certificate request.

Syntax

```
uint32 JNH_inquire_certreq_privkey_EE(uint32 reqId,
                                      utf8String * algorithm,
                                      uint32 * length)
```

Parameters

reqId – input
The identifier of the certificate request.

algorithm – output
The algorithm associated with the private key, either id_dsa or id_rsa.

length – output
The length of the private key in bits, typically 1024.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_privkey_EE

Example

```
uint32 length;
uint32 reqId;     // id of certificate request
utf8String algorithm;
retVal = JNH_inquire_certreq_privkey_EE (reqId, &algorithm, &length);
```

JNH_inquire_certreq_serialnumber

Retrieves the serial number from a certificate request.

Syntax

```
uint32 JNH_inquire_certreq_serialnumber(uint32 reqId,
                                        octetString * serialNumber)
```

Parameters

reqId – input
The identifier of the certificate request.

serialNumber – output
The serial number of the certificate, a long integer stored in the octetString data buffer. This buffer needs to be cast to long before it can be used.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The serial number for a request is only available after a certificate has been created from the request.

Related Functions

- JNH_set_revreq_certserialnumber

Example

```
uint32 reqId;     // id of certificate request
uint32 rc;
octetString serialNumber
serialNumber.data = NULL;
rc = JNH_inquire_certreq_serialNumber(reqId, &serialNumber);
if (serialNumber.data !=NULL)
printf("Serial Number was %x\n", * (uint32*)(serialNumber.data));
```

JNH_inquire_certreq_startdate

Retrieves the starting date (the ″valid not before″ attribute) from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_startdate(uint32 reqId,
                                     utcDateTime * startdate)
```

Java

```
int JAVA_inquire_certreq_startdate(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

startdate – output
The starting date of the request.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_enddate
- JNH_set_certreq_startDate

Example

C++

```
uint32 reqId;     // id of certificate request
utcDateTime startdate;
uint32 rc;
rc = JNH_inquire_certreq_startdate(reqId, &startdate);
```

Java

```
Date StartDate;
int retVal;
int reqId;     // id of certificate request
retVal = jonahInterface.JAVA_inquire_certreq_startdate(reqId);
if (retVal == 0)
{
   startdate.year = jonahInterface.year;
   startdate.month = jonahInterface.month;
   startdate.day = jonahInterface.day;
}
```

JNH_inquire_certreq_status

Retrieves the status of a certificate request.

Syntax

```
uint32 JNH_inquire_certreq_status(uint32 reqId, uint32 * status)
```

Parameters

reqId – input
The identifier of the certificate request.

status – output
The status of the certificate request. See the Objstates.h file for the status definition.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_get_object_state

Example

```
uint32 reqId;     // id of certificate request
uint32 status
uint32 rc;
rc = JNH_inquire_certreq_status(reqId, &status);
```

JNH_inquire_certreq_subject

Retrieves the subject name from a certificate request.

Syntax

C++

```
uint32 JNH_inquire_certreq_subject(uint32 reqId,
                                   utf8String * subject)
```

Java

```
int JAVA_inquire_certreq_subject(int reqId)
```

Parameters

reqId – input
The identifier of the certificate request.

subject – output
The subject name of the certificate request, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_certreq_subject

Example

C++

```
utf8String subject;
uint32 rc;
uint32 reqId;     // id of certificate request
rc = JNH_inquire_certreq_subject(reqId, &subject);
if (rc == 0)
{
   printf ("Subject Name = %s\n", subject);
}
```

Java

```
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_inquire_certreq_subject(reqId);
if (retVal == 0)
{
   String subjName = jonahInterface.retStr;
}
```

JNH_inquire_certreq_subjectkey_algorithm

Retrieves the subject key algorithm from a certificate request.

Syntax

```
uint32 JNH_inquire_certreq_subjectkey_algorithm(uint32 reqId,
                                                octetString * algorithm)
```

Parameters

reqId – input
The identifier of the certificate request.

algorithm – output
The certificate request's subject key algorithm in the numeric OID dot-format.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

```
uint32 reqId;     // id of certificate request
octetString algorithm;
algorithm.data = NULL;
algortihm.length = 0;
uint32 rc;
rc = JNH_inquire_certreq_subjectKey_algorithm(reqId, &algorithm);
```

JNH_inquire_revreq_certIssuer

Returns the name of the issuer of a specified revocation request.

Syntax

C++

```
uint32 JNH_inquire_revreq_certIssuer(uint32 reqId,
                                     uint32 index,
                                     utf8String * certIssuer)
```

Java

```
int JAVA_inquire_revreq_certIssuer(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request being queried. This identifier must be type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

certIssuer – output
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7. This parameter must be passed in as NULL. On output, the field will point to malloc’ed memory, which the caller is responsible for freeing.

Usage

CA, EE, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_revreq_certIssuer

Example

C++

```
utf8String resultIssuer = NULL;
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_inquire_revreq_certIssuer(reqId, (uint32)0, resultIssuer);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_inquire_revreq_certIssuer(reqId);
if (retVal == 0)
{
   String issuerName = jonahInterface.retStr;
}
```

JNH_inquire_revreq_certserialnumber

Returns one of the certificate serial numbers from a revocation request.

Syntax

C++

```
uint32 JNH_inquire_revreq_certserialnumber(uint32 reqId,
                                           uint32 index,
                                           octetString * serialNumber)
```

Java

```
int JAVA_inquire_revreq_certserialnuamber(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request being queried. This identifier must be of type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

serialNumber – output
The serial number of the certificate, a long integer stored in the octetString data buffer. This buffer needs to be cast to long before it can be used.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_revreq_certserialnumber

Example

C++

```
octetString ser;
ser.data = NULL;
ser.length = 0;
uint32 rc;
uint32 reqId;     // id of the revocation request to be queried
rc = JNH_inquire_revreq_certserialnumber(reqId, (uint32)0, &ser);
```

Java

```
int reqId;     // id of the revocation request to be queried
int retVal;
retVal = jonahInterface.JAVA_inquire_revreq_certserialnumber(reqId);
if (retVal == 0)
{
   String ser = jonahInterface.retStr;
}
```

JNH_inquire_revreq_certserialnumbers

Returns a list of the certificate serial numbers for all the certificates being revoked by this request.

Syntax

```
uint32 JNH_inquire_revreq_certserialnumbers(uint32 reqId,
                                            uint32 * num_of_certs,
                                            octetString serialNumber[])
```

Parameters

reqId – input
The identifier of the revocation request being queried. This identifier must be of type ObjClTypeRev.

num_of_certs – input/output
The number of certificates in the revocation request.

If the serialNumber parameter is set to NULL, the number of certificates will be written to num_of_certs. If the serialNumber parameter is not NULL, set the num_of_certs input parameter to represent the size of the serialNumber array.

On output, this parameter contains the number of certificate serial numbers that were written to the array.

serialNumber[] – output
The array of certificate serial numbers whose revocation is being requested, in the standard ASN.1 binary format for integers. These are variable-length, highest-order-first (big-endian) binary integers with redundant leading 0 or FF bytes removed. A leading 0 byte is considered redundant if the next byte's value is less than 128. A leading FF byte is considered redundant if the next byte's value is greater than or equal to 128.

On input, set the data fields of the octetString to NULL.

On output, the data fields point to malloc'ed storage, which the caller is responsible for freeing.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_revreq_certserialnumber
- JNH_set_revreq_certserialnumber

Example

```
octetString * a = NULL;
uint32 asize = 0;
uint32 numberofRequest = 6;
asize = numberofRequests * sizeof(octetString);
octetString a = (octetString *) malloc (asize);
uint32 tempnum = numberOfRequests;
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_inquire_revreq_certserialnumbers(reqId, &tempNum, a);
```

JNH_inquire_revreq_hold_instruction_code

Returns the hold_instruction_code from one of the certificates being revoked in a revocation request.

Syntax

```
uint32 JNH_inquire_revreq_hold_instruction_code(uint32 reqId,
                                                uint32 index,
                                                octetString * code)
```

Parameters

reqId – input
The identifier of the revocation request being queried. This identifier must be of type ObjClTypeRev.

Index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

code – output
The object ID of the hold instruction code, in the standard ASN.1 binary format for object identifiers.

Note: Since hold_instruction_code is only properly used when a certificate has been placed in “certificateHold” status, it is common for no hold_instruction_code to be present in a revocation request. In such cases, no code is returned despite a successful return value (0), the data pointer is set to NULL and the length set to 0.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_revreq_hold_instruction_code

Example

```
octetString outHold;
outHold.data = NULL;
outHold.length = 0;
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_inquire_revreq_hold_instruction_code(reqId, (uint32)0, &outHold);
```

JNH_inquire_revreq_invalidityDate

Returns the invalidity date from one of the certificates being revoked in a revocation request.

Syntax

C++

```
uint32 JNH_inquire_revreq_invalidityDate(uint32 reqId,
                                         uint32 index,
                                         utcDateTime * invalidityDate)
```

Java

```
int JAVA_inquire_revreq_invalidityDate(int reqId)
```

Parameters

reqId or reqId – input
The identifier of the revocation request being queried. The identifier must be of type ObjClTypeRev.

Index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

invalidityDate – output
The invalidity date of the requested certificate.

Note: This is an optional extension to the CRL entry. Because of this, it is possible for no data to return, despite a successful return value (0). In such a case, all fields will be set to 0.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The invalidity date is the date that the certificate is suspected to have been compromised and might be earlier than the actual revocation date.

Related Functions

- JNH_set_revreq_invalidityDate

Example

C++

```
uint32 rc;
uint32 reqId;     // id of revocation request
utcDateTime invalidDate;
rc = JNH_inquire_revreq_invalidityDate(reqId, (uint32)0, &invalidDate);
```

Java

```
int retVal;
int reqId;    // id of revocation request
retVal = jonahInterface.JAVA_inquire_revreq_invalidityDate(reqId);
if (retVal == 0)
{
   System.out.println ("Invalidity date= " + jonahInterface.month + "/"
   + jonahInterface.day + "/" + jonahInterface.year);
}
```

JNH_inquire_revreq_reason

Returns the reason code from one of the certificates being revoked in a revocation request.

Syntax

C++

```
uint32 JNH_inquire_revreq_reason(uint32 reqId,
                                 uint32 index,
                                 int * reasonFlags)
```

Java

```
int JAVA_inquire_revreq_reason(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request being queried. The identifier must be of type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

reasonFlags – output
The reason code from the requested certificate.

The reason codes are as follows:

0	REV_REASON_NONE
1	REV_REASON_UNUSED
2	REV_REASON_KEY_COMPROMISE
4	REV_REASON_CA_COMPROMISE
8	REV_REASON_AFFILIATION_CHANGED
16	REV_REASON_SUPERSEDED
32	REV_REASON_CESSATION_OF_OPERATION
64	REV_REASON_CERTIFICATE_HOLD

For a complete list of the revocation reason codes and their definitions, see Include\ASN1\X509.h REASON_*.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_set_revreq_reason

Example

C++

```
int outReasons = 0;
uint32 reqId;     // id of revocation request
uint32 rc;
rc = JNH_inquire_revreq_reason(reqId, (uint32) 0, &outReasons);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_inquire_revreq_reason(reqId);
if (retVal == 0)
{
   int outReasons = jonahInterface.retInt;
}
```

JNH_inquire_revreq_requests

Determines how many certificates are being revoked by the revocation request.

Syntax

```
uint32 JNH_inquire_revreq_requests(uint32 reqId,
                                   uint32 * nReqs)
```

Parameters

reqId – input
The identifier of the revocation request. The identifier must be of type ObjClTypeRev.

nReqs – output
Returns the number of certificates being revoked by this revocation request.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

```
uint32 nReqs;
uint32 rc;
uint32 reqId;     // id of the revocation request
rc = JNH_inquire_revreq_requests(reqId, &nReqs);
```

JNH_Keypair_Selected

Allows the EE to select a key pair from the list returned by JNH_List_Existing_Keypairs and use that key pair in a new certificate request.

Syntax

```
uint32 JNH_Keypair_Selected(const char * ppin,
                            char * pjkeyID)
```

Parameters

ppin – input
A pointer to the smart card user PIN.

pjKeyID – input
The identifier of the key pair to be used in the new certificate request.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_List_Existing_Keypairs

Example

```
const char * ppin;
char * pjkeyID;

rv = JNH_KeyPair_Selected(ppin, pjkeyID);
```

JNH_keystore_inquire_cert

Retrieves a certificate from the key store.

Syntax

C++

```
uint32 JNH_keystore_inquire_cert(const utf8String entryName,
                                 const utf8String pin,
                                 const octetString keyID,
                                 octetString * cert_buffer);
```

Java

```
int JAVA_keystore_inquire_cert(String entryName,
                               Vector keyID,
                               String pin)
```

Parameters

entryName – input
The subject name of the certificate to retrieve from the key store.

pin – input
The security person identification number to access the key store.

keyID – input
The key identifier of the certificate to retrieve from the key store.

cert_buffer – output
A buffer containing the DER encoding of the certificate retrieved from the key store.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_list_SC_certs

Example

C++

```
uint32 retVal;
octetString keyID;
utf8String entryName;
utf8String pin;
octetString cert_buffer;
retVal = JNH_keystore_inquire_cert(entryName, pin, keyID, cert_buffer);
```

Java

```
String pin = "SOPIN";
retVal = jonahInterface.JAVA_list_SC_certs(pin);

if (retVal != 0)
```

```
{

      //KEYSTORE Error Occurred retrieving the certs list.
}
//Return Vector object with all the certs info.
jonahInterface.retVector;

entryName = (String) jonahInterface.retVector.elementAt(1);

keyIDVector = (java.util.Vector) jonahInterface.retVector.elementAt(1);

retVal = frameMain.jonahInterface.JAVA_keystore_inquire_cert(entryName,
                                                             keyIDVector,
                                                             pin);

//byte[] that contains the DER Encoding of the certificate.
frameMain.jonahInterface.retByteArr;
```

JNH_List_Existing_Keypairs

Lists any existing public and private key pairs stored on a smart card.

Syntax

```
uint32 JNH_List_Existing_Keypairs(const char * ppin,
                                  uint32 & retNumKeys,
                                  octetString * keyIDs,
                                  uint32 * algorithmID,
                                  uint32 * keyUsage)
```

Parameters

ppin – input
A pointer to the smart card user PIN.

retNumKey – output
The number of keys found on the smart card.

keyIDs – output
The unique identifier of the key pair.

algorithm – output
The algorithm supported by the key pair.

keyUsage – output
The key pair's intended usage.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API allows the EE to use a pre-existing key when generating a new certificate request.

Related Functions

- JNH_Keypair_Selected

Example

```
const char * ppin;
uint32 * retNumKeys;
octetString * keyIDs;
uint32 * pAlgorithmID;
uint32 * pKeyUsage;

rv = JNH_List_Existing_Keypairs(ppin, retNumKeys, keyIDs, pAlgorithmID, pKeyUsage);
```

JNH_list_objects

Causes all active objects to announce themselves.

Syntax

C++

```
uint32 JNH_list_objects()
```

Java

```
int JAVA_list_objects()
```

Parameters

None.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This routine allows the GUI to synchronize any records it maintains with the actual contents of the active object store. Since objects in the object store might persist across server invocations, the GUI should call this routine on initialization to determine the state of the object store.

Related Functions

- JNH_list_surrogates

Example

C++

```
JNH_list_objects();
```

Java

```
int retVal = jonahInterface.JAVA_list_objects();
```

JNH_list_SC_certs

Returns the subject names and key identifiers for all the certificates in the key store.

Syntax

C++

```
uint32 JNH_list_SC_certs(const char *pin,
                         uint32 & numCerts,
                         unsigned char *** keyIds,
                         uint32 ** keyIdLengths,
                         unsigned char *** certNames,
                         uint32 ** certNameLengths)
```

Java

```
int JAVA_list_SC_certs(String pin)
```

Parameters

pin – input
The security officer or user private identification number (PIN) to access the key store.

numCerts – output
The number of certificates in the key store.

keyIds – output
An array of the key identifiers for each certificate in the key store.

keyIdLengths – output
An array of the lengths of each certificate's key identifier.

certNames – output
An array of the subject names for all the certificates in the key store.

certNameLengths – output
An array of the length of each certificate's subject name.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_keystore_inquire_cert

Example

C++

```
uint32 retVal;
uint32 retNumCerts;
char * pin;
unsigned char ** keyIds;
uint32 * keyIdLengths;
unsigned char ** certNames;
```

```
uint32 * certNameLengths;

retVal = JNH_list_SC_certs(pin, retNumCerts, &keyIds, &keyIdLengths, &certNames,
&certNameLengths);
```

Java

```
String pin = "SOPIN";
retVal = jonahInterface.JAVA_list_SC_certs(pin);

if (retVal != 0)
{

      //KEYSTORE Error Occurred retrieving the certs list.
}
//Return Vector object with all the certs info.
jonahInterface.retVector
```

JNH_list_surrogates

Causes all surrogate objects to announce themselves.

Syntax

```
uint32 JNH_list_surrogates()
```

Parameters

None.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This routine allows the GUI to synchronize any records it maintains with the actual contents of the active object store. Since objects in the object store might persist across server invocations, the GUI should call this routine on initialization to determine the current state of the object store.

Related Functions

- JNH_list_objects

Example

```
uint32 rc;
rc = JNH_list_surrogates();
```

JNH_modify_certreq_extension

Modifies an extension in a certificate request.

Syntax

C++

```
uint32 JNH_modify_certreq_extension(uint32 xtype,
                                    uint32 reqId,
                                    utf8String extId,
                                    octetString value,
                                    IBOOL critical)
```

Java

```
int JAVA_modify_certreq_extension(int xtype,
                                  int reqId,
                                  String extId,
                                  String value,
                                  int critical)
```

Parameters

xtype – input
The type of extension to be modified:

1	HOST_ID_MAPPING
2	KEY_USAGE
3	SUBJECT_ALT_NAME
4	PRIVATE

reqId – input
The identifier of the certificate request.

extId – input
The identifier of the extension to modified in the certificate request. This parameter is primarily used for private extensions, where it is provided in a dot string format (such as 1.4.5.12).

value – input
The octetstring value of the extension.

critical – input
Determines whether or not the extension is critical:

0	no
1	yes

Note: Private extensions must not be marked critical.

Usage

RA, CA, EE

Remarks

Only one Key Usage and Subject Alternative Name extension identifier can be added to each certificate. A host id mapping extension can be added for each unique host name, and a private extension can be added for each unique extension identifier.

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_add_certreq_extension

Example

C++

```
uint32 rc;
uint32 xtype;
uint32 reqId;              // id of certificate request
uint32 critical= 0;
octetString value;         // specify the new value for the extension
utf8String extId = "host1";
xtype = HOST_ID_MAPPING;
status = JNH_modify_certreq_extension(xtype, reqId, extId, value, critical);
```

Java

```
int retVal;
int xtype = HOST_ID_MAPPING;
int reqId;     // id of certificate request
String extId= "host1";
String value= "/host1/id1/";
int critical = 1;
retVal = jonahInterface.JAVA_modify_certreq_extension(xtype,
         reqId, extId, value, critical);
```

JNH_new_revreq

Creates a new revocation request and returns the revocation request's identifier.

Syntax

```
uint32 JNH_new_revreq(const octetString *serialNumber,
                      const utf8String issuer,
                      uint32 * reqId)
```

Parameters

serialNumber – input
The serial number of the certificate to be revoked.

issuer – input
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in "Parameter format" on page 7.

reqId – output
The identifier of the new revocation request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_revreq

Example

```
char sN[]= "7777";
octetString serialNumber;
uint32 rc;
serial.data = (unsigned char*) &sN;
serial.length = strlen(sN);
uint32 reqId;     // id of revocation request
rc = JNH_new_revreq(&serialNumber, (utf8String) "/CN=ISSUER", &reqId);
```

JNH_pkcs12scExportFile

Exports a private key and public certificate from a smart card.

Syntax

C++

```
JNH_pkcs12scExportFile(const octetString &key_id.
                       const octetString &extdShortName,
                       const utf8String p12_password,
                       const utf8String sc_password,
                       const octetString &algorithmId,
                       octetString * exportfile,
                       uint32 iteration_count,
                       const uint32 slot_id)
```

Java

```
public native int JAVA_PKCS12_Export_from_Keystore(String entryName,
                                                   Vector keyID,
                                                   String filePassword,
                                                   String keystorePassword,
                                                   int algorithmID,
                                                   int iterationCount)
```

Parameters

key_id – input
The identifier of the private key to export.

extdShortName – input
The extended short name from the smart card's certificate structure.

p12_password – input
A password for the exported file. This password should be known by both the sender and receiver of the file.

sc_password – input
The password to the slot on the smart card where the file is stored.

algorithmId – input
The identifier of the type of algorithm used to encrypt the file. Can be one of the following:

RC2_40BIT_KEY	40-bit encryption algorithm
RC2_128BIT_KEY	RSA 128-bit encryption algorithm
DES3_3KEY	192-bit triple Data Encryption Standard (DES) encryption.

exportfile – output
A flat file containing DER encodings of the private key and public certificate.

This file must be saved as binary.

You must free the export file after this call.

iteration_count – input
The number of times to encrypt the file. The default and lowest valid value is 1.

slot_id – input
The identifier for the smart card slot that contains the private key and public certificate.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h and crmsg.h files for details.

Remarks

PKCS #12 specifies a portable format for storing or transporting personal information. The Trust Authority version of PKCS #12 restricts the information that can be exported to private keys and public certificates. Under the current implementation, certificate chains cannot be exported.

To use the PKCS #12 APIs, you must include in your application, using the #include statement, the Jonah.h and JonahAlg.h files.

The key_id, extdShortName, p12_password, sc_password, and algorithmId parameters must not be NULL.

Related Functions

- JNH_pkcs12UserExportFile
- JNH_pkcs12ImportFile

Example

C++

```
uint32  status;
int serverType; //0 ->EE, 1 ->RA, 2 ->CA
int algType; //algorithm type: 1 -> for RC2_40BIT_KEY
char uPin[128]; //user pin
char p12Pin[128]; //pksc12 pin to export the file.
//When this file is imported, it need this pin again
char Path[128];  //file name to save the cert exported from the smartcard

.
.
.

int exportToFile(int keyIdLength, unsigned char *keyId, int certNameLength,
unsigned char *certName)
{
octetString key, n, file, alg;
char t[128]= "";
strcat(t, Path);    //prepare a file to store the export file
strcat(t, (char *)certName);
strcat(t, ".cert");

key.data = keyId;
key.length = keyIdLength;
n.data = certName;
n.length = certNameLength; // strlen((char *)n.data);


//algType = 1 -> for RC2_40BIT_KEY
if (algType == 1)
{
// alg.data = new unsigned char[strlen((char *) RC2_40BIT_KEY)];
alg.length = strlen((char *) RC2_40BIT_KEY);
alg.data = RC2_40BIT_KEY;

}


status = JNH_pkcs12ScExportFile(key, n, (utf8String) p12Pin, (utf8String) uPin,
 (octetString) alg, &file, 1, 1);
if (status)
{
printf("Error calling JNH_pkcs12scExportFile() with status = %d\n",status);
return -1;
}


if ((fp = fopen(t, "wb")) == NULL)
{
printf("Can not open file %s to store the cert.\n",t);
return -1;
}
//write the cert to a file
fwrite((char *)file.data, (int)file.length, 1, fp);
fclose(fp);

return 0;
}

.
.
.

int performExport()
{

uint32 retNumCerts;
unsigned char ** keyIds;
uint32 * keyIdLengths;
unsigned char ** certNames;
uint32 * certNameLengths;

//start the server
status = JNH_start_server(serverType);
if (status)
{
printf("Error calling JNH_start_server() with status = %d\n",status);
return -1;
}
status = JNH_server_login_pwd((utf8String) uPin);
if (status)
{
printf("\nError calling JNH_server_login_pwd() with status %d\n", status);
return -1;
}
status = JNH_register_callbacks( &notify, &display);
if (status)
{
printf("\nError calling JNH_register_callbacks() with status %d\n", status);
return -1;
}

status = JNH_list_SC_certs(uPin,retNumCerts,&keyId,&keyIdLengths,&certNames,
 &certNameLengths);
if (status)
{
printf("Error calling JNH_list_SC_certs() with status = %d\n",status);
return -1;
}
```

```
if (retNumCerts < 1)
{
printf("There is no cert in the smartcart.\n");
return -1;
}
printf("==================\n");
printf("\nNumber of cert is %d\n",retNumCerts);
for (int i = 0; i < (int)retNumCerts; i++)
{
printf("\nCert  %d:\n",i);
// printf("Key Id is %s\n",keyIds[i]);
printf("CertName is %s\n\n",certNames[i]);
}
printf("==================\n");


{
int start, stop, j;
bool valid = false;
while (!valid)
{
printf("With a list of certs above, enter a START number of the cert you
       want to export: ");
cin >> start;
cout << endl;
printf("With a list of certs above, enter a STOP number of the cert you
       want to export: ");
cin >> stop;
cout << endl;
if (((start < (int)retNumCerts) && (start >= 0)) && ((stop < (int)retNumCerts)
       && (stop >= start)))
valid = true;
else
printf("\nInvalid value!!! Try again.\n");

}

for (j = start; j <= stop; j++)
{
if (exportToFile(keyIdLengths[j], keyIds[j], certNameLengths[j], certNames[j])
        == -1)
{
printf("Error calling exportToFile()\n");
return -1;
}
}

//------------

return 0;
}

   ⋮
```

Java

```
String entryName ; //Entry name for the Certificate to be Exported.
Vector keyID;     //Vector that contains the Key Id for the Certificate
                  //to be Exported.

String keystorePassword = "PIN";
String filePassword = "filePW";
int algorithmID = 1;
int iterationCount = 1;

frameMain.jonahInterface.JAVA_PKCS12_Export_from_Keystore(entryName,
                                                           keyID,
                                                           filePassword,
                                                           keystorePassword,
                                                           algorithmID,
                                                           iterationCount);

if (retVal == 0) {

            byte[] fileBuffer = frameMain.jonahInterface.retByteArr;

}
```

JNH_pkcs12UserExportFile

Exports a private key and public certificate without first retrieving them from the smart card.

Syntax

```
JNH_pkcs12UserExportFile(const octetString &key_id.
                         const utf8String p12_password,
                         octetString &priv_key,
                         octetString &pub_cert,
                         const octetString &algorithmId,
                         octetString * exportfile,
                         uint32 iteration_count)
```

Parameters

key_id – input
The identifier of the private key to export to the smart card.

p12_password – input
A password for the exported file. This password should be known by both the sender and receiver of the file.

priv_key – input
The flattened DER encoding of the private key. Create the octetString.data structure by performing a privateKeyInfo.write on the private key.

pub_cert – input
The flattened DER encoding of the public certificate. Create the octetString.data structure by performing an x.509.write on the public certificate.

algorithmId – input
The identifier of the type of algorithm used to encrypt the file. Can be one of the following:

RC2_40BIT_KEY	40-bit encryption algorithm
RC2_128BIT_KEY	RSA 128-bit encryption algorithm
DES3_3KEY	192-bit triple Data Encryption Standard (DES) encryption

exportfile – output
A flat file containing DER encodings of the private key and public certificate.

This file must be saved as binary.

You must free the export file after this call.

iteration_count – input
The number of times to encrypt the file. The default and lowest value is 1.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h and crmsg.h files for details.

Remarks

PKCS #12 specifies a portable format for storing or transporting personal information. The Trust Authority version of PKCS #12 restricts the information that can be exported to private keys and public certificates. Under the current implementation, certificate chains cannot be exported.

To use the PKCS #12 APIs, you must include in your application, using the #include statement, the Jonah.h and JonahAlg.h files.

The key_id, p12_password, priv_key, pub_cert, and algorithmId parameters must not be NULL.

Related Functions

- JNH_pkcs12scExportFile
- JNH_pkcs12ImportFile

Example

```
x509_certificate    publicCert;           // A complete public certificate
octetString         flat_privKeyInfo,     // DER encoded private key info structure
                    flat_cert,            // DER encoded public certificate
                    algorithmId,          // Well known IDs are found in JonahAlg.h
                    key_id,               // Key ID that associates the private key
                                          // with the corresponding public certificate.
                    flatp12_exportfile;   // PFX PDU DER encoded file

utf8String          p12_password;         // Password that is used to encrypt data
                                          // of the PKCS12 exportfile.

uint32              status           = 0, // API return status code
                    iteration_count  = 1; // Number of encryption iterations on each
                                          // encryption call.
buffer_t            certBuf;              // Temp output buffer for flatting the x509 certificate

.
.

/* This example leaves the environment initialization to the sample code user */

.
.
.
.

/*This example leaves the retrival of input parameters to the user. (i.e. private
key, public cert, keypair keyid, etc. */

.
.
.
.

algorithmId.data = RC2_40BIT_KEY;
algorithmId.length = strlen((char *)RC2_40BIT_KEY);

status = publicCert.write(certBuf);
if(status) return status;

flat_publicCert.data = certBuf.data;
flat_publicCert.length = certBuf.data_len;
```

```
status = JNH_pkcs12UserExportFile(temp_key_id,
                                  p12_password,
                                  flat_privKeyInfo,
                                  flat_publicCert,
                                  algorithmId,
                                  &flatp12_exportfile,
                                  iteration_count);
if(status) return status;

.
.
.
```

JNH_pkcs12ImportFile

Imports a private key and public certificate to the smart card.

Syntax

C++

```
uint32 JNH_pkcs12ImportFile(const octetString importFile,
                            const utf8String p12_password,
                            const utf8String sc_password,
                            octetString * keypair_id,
                            octetString * extd_shortName,
                            const uint32 slot_id)
```

Java

```
public native int JAVA_PKCS12_Import_to_Keystore(byte[] importFile,
                                                 String filePassword,
                                                 String keystorePassword)
```

Parameters

importFile – input
The identifier of the file to import.

p12_password – input
A password for the imported file. This password should be known by both the sender and receiver of the file.

sc_password – input
The password to the slot on the smart card where the file will be stored.

keypair_id – output
The identifier of the private key and public certificate successfully imported.

After the call, you must free the keypair_id.

extd_shortName – output
The extended short name of the certificate imported to the smart card.

After the call, you must free the extd_shortName.

slot_id – input
The smart card slot where the file will be stored.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h and crmsg.h files for details.

Remarks

PKCS #12 specifies a portable format for storing or transporting personal information. The Trust Authority version of PKCS #12 restricts the information that can be exported to private keys and public certificates. You can import a certificate chain to Trust Authority; however, Trust Authority will only store the user's certificate (the leaf certificate).

To use the PKCS #12 APIs, you must include in your application, using the #include statement, the Jonah.h and JonahAlg.h files.

Related Functions

- JNH_pkcs12scExportFile
- JNH_pkcs12UserExportFile

Example

C++

```
uint32  status;
int serverType;      //0 ->EE, 1 ->RA, 2 ->CA
int algType;         //algorithm type: 1 -> for RC2_40BIT_KEY
char uPin[128];          //user pin
char p12Pin[128];        //pksc12 pin to export the file.
                        //When the file is imported, it needs this pin again
char fileName[128];
char Path[128];
FILE *fp;

.
.
.

int performImport()
{
char buf[128];
//start the server
status = JNH_start_server(serverType);
if (status)
{
printf("Error calling JNH_start_server() with status = %d\n",status);
return -1;
}
status = JNH_server_login_pwd((utf8String) uPin);
if (status)
{
printf("\nError calling JNH_server_login_pwd() with status %d\n", status);
return -1;
}
status = JNH_register_callbacks( &notify, &display);
if (status)
{
printf("\nError calling JNH_register_callbacks() with status %d\n", status);
return -1;
}

//----------
//try to read the extented shortname out from the file
ifstream inf(fileName);

if (!inf)
{
cout << "Cannot open file " << fileName << endl;
return -1;

}

while(!inf.eof())
{
char f[256];
unsigned char *info;
int  len;
octetString importinfo, keypair, esname;
```

```
inf.getline(buf, sizeof(buf));
if (strcmp(buf, "") == 0)
break;

cout << "\nExtended short name is " << buf << endl;
//prepare the file to go get the cert info from the corresponding file
strcpy(f, ""); //clear buffer f before strcat with any other string
strcat(f, Path);
strcat(f, buf);
strcat(f, ".cert");

//Read the cert info from the exported file
if ( ReadMessage(f, &info, &len) != 0)
{
printf("Error calling ReadMessage()\n");
return -1;
}

//set up parameter to import the file
importinfo.data = info;
importinfo.length =len;

status = JNH_pkcs12ImportFile(importinfo, (unsigned char *)p12Pin,
    (unsigned char *)uPin, &keypair, &esname, 1);
if (status)
{
printf("Error calling JNH_pkcs12ImportFile() with status = %d\n",status);
return -1;
}
if (esname.data != NULL)
{
// cout << "Extended short name value return from inport file is"
//        << esname.data <
// cout << "Extended short name value return from inport file is"
//        << esname.data <
```

Java

```
byte [] bytesBuffer = {...};

String keystorePassword = "PIN";
String filePassword = "filePW";

int retVal = frameMain.jonahInterface.JAVA_PKCS12_Import_to_Keystore
(bytesBuffer,filePassword,keystorePassword);

if (retVal == 0){
// Import Action was successfull.
}
```

JNH_preregister_crosscert

Creates a cross-certification record for a CA, using the preregistration record created by JNH_RA_preregister_crosscert.

Syntax

```
uint32 JNH_preregister_crosscert(const utf8String PreRegistrationRecord,
                                 const utf8String passwd,
                                 uint32 *ReqId)
```

Parameters

PreRegistrationRecord – input
A record that contains the template information set by the RA, including the CA name, RA name, and RA URL. This template record is created by the RA by a call to JNH_RA_preregister_crosscert and is stored in a file that the subject CA can open while cross-certifiying.

passwd – input
″0″

ReqId – output
The identifier of the cross-certification request created in the CA’s object store.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_RA_preregister_crosscert

Example

```
if (ReadMessage("c:/ccertreg.reg", &buf, &bufLen) != 0) {
   exit(-1);
}

msg = (utf8String) malloc(bufLen +1);
memset(msg, 0, bufLen + 1);
memcpy(msg, buf, bufLen);

if ((status = JNH_preregister_crosscert(msg, (utf8String) "TEST", &id))) {
   printf("objid = %d\n", id);
   printf("prereg status = %d\n", status);
   return -1;
}
```

JNH_preregister_user

Validates a preregistration record and creates the object store entry for the certificate.

Syntax

C++

```
uint32 JNH_preregister_user(const utf8String PreRegistrationRecord,
                            const utf8String Password,
                            uint32 * reqId)
```

Java

```
int JAVA_preregister_user(String CAName,
                          String UserName,
                          int Expires,
                          String Password)
```

Parameters

PreRegistrationRecord – input
A record that contains the template information set by the RA, including the CA name, RA name, and RA URL. This template record is created by the RA by a call to JNH_RA_preregister_user and is stored in a file that the EE can open while pre-registering.

Password – input
″0″

ReqID – output
The object identifier created for this request in the object store.

CAName – input
The name of the CA to which the request will be sent.

UserName – input
The name of the EE.

Expires – input
The expiration date of the certificate.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_RA_preregister_User

Example

C++

```
uint32 reqId;
uint32 status;
utf8String PreRegistrationRecord;
```

```
status = JNH_RA_preregister_user("/C=US", "CN=A_USER",0,
         (utf8String) "rd_pw", &PreRegistrationRecord);
status = JNH_preregister_user(PreRegistrationRecord,
         (utf8String) "user_pw", &reqId);
```

Java

```
jonahInterface.JAVA_RA_preregister_user("/C=US", "CN=A_USER",0, "rc_pw");
String PreRegistration Record = jonahInterface.retStr;
int retVal = jonahInterface.JAVA_preregister_user
                (PreregistrationRecord, "user_pw");
int reqId = jonahInterface.retInt;
```

JNH_publish_certificate

Publishes the certificate.

Syntax

```
uint32 JNH_publish_certificate(uint32 reqID)
```

Parameters

reqID – input
The identifier of the certificate request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

Depending on the configuration, publication may involve storing the certificate in a directory, returning it to the requester, or both.

Related Functions

- JNH_create_certificate

Example

```
uint32  rc;
uint32  reqId;

rc = JNH_publish_certificate(reqId);
```

JNH_publish_CRL

Publishes a CRL to the RA.

Syntax

```
uint32 JNH_publish_CRL(uint32 crlId)
```

Parameters

crlId – input
The identifier of the CRL to publish.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_CRL
- JNH_request_CRL

Example

```
uint32 JNH_publish_CRL(uint32 crlId)
{
   jnh_api_call apiCall;
   jnh_api_ret  apiRet;
   uint32       rc;
   long         lrc;

   apiCall.apiIndex.set_value(JNH_PUBLISH_CRL);
   apiCall.params.int1.value.set_value(crlId);
   apiCall.params.int1.set_optional(false);

   if(getReturn(apiCall,apiRet)) return API_NO_RESPONSE;

   apiRet.ret.returnCode.get_value(lrc);
   rc = lrc;

   return rc;
}
```

JNH_RA_nonpkix_create_revreq

Creates a revocation request object for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_nonpkix_create_revreq(const octetString *serialNumber,
                      const utf8String issuer,
                      uint32 *reqId));
```

Parameters

serialNumber – input
A pointer to an octetString structure that contains the serial number of the certificate to be revoked.

issuer – input
The Certificate Authority that issued this certificate, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

reqId – input/output
A pointer to the integer where the request ID (also known as object store ID) of the certification request being revoked is returned.

Usage

RA

Return Values

0	Normal, successful completion
>0	An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_nonpkix_request_revocation

Example

```
char sn[] = "1234";
uint32 reqId;
uint32 status=0;
octetString serialNumber;
serialNumber.data = &sn;
serialNumber.length = strlen(sn);
utf8String issuer="/C=us/O=IBM/OU=MyOrg";
uint32 JNH_RA_nonpkix_create_revreq(&serialNumber, issuer, &reqId);
```

JNH_RA_nonpkix_request_revocation

Requests a revocation for a certificate for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_nonpkix_request_revocation(uint32 reqId));
```

Parameters

reqId – input
The request ID (also known as object store ID) of the certificate being revoked.

Usage

RA

Return Values

0 Normal, successful completion

>0 An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_nonpkix_create_revreq

Example

```
uint32 reqId;
unit32 status = 0;
status = JNH_RA_nonpkix_request_revocation(reqId);
```

JNH_RA_nonpkix_user_check

Checks on the status of a registration request for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_nonpkix_user_check(uint32 reqId,
                                 uint32 * RequestStatus)
```

Parameters

reqId – input
The identifier of the initialization record created by JNH_RA_preregister_nonpkix_user.

RequestStatus – output
The status of the registration request:

1	JNH_CertRequestSubmitted
2	JNH_CertRequestApproved
3	JNH_CertRequestRejected
4	JNH_CertRequestError

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_preregister_nonpkix_user

Example

```
uint32 reqId;              // request identifier
uint32 RequestStatus       // request status
uint32 status;
status = JNH_CA_nonpkix_user_check(reqId, &RequestStat);
```

JNH_RA_nonpkix_user_done

After the non-PKIX user registration is complete, this routine cleans up the object store entry.

Syntax

```
uint32 JNH_RA_nonpkix_user_done(uint32 reqId)
```

Parameters

reqId – input
The identifier of the initialization request (created by JNH_RA_register_nonpkix_user) to be removed from the object store.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_preregister_nonpkix_user

Example

```
uint32 reqId;         // request identifier
uint32 status;
status = JNH_RA_nonpkix_user_done(reqId);
```

JNH_RA_nonpkix_user_get_cert

Retrieves the certificate for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_nonpkix_user_get_cert(uint32 reqId,
                                    octetString ** Certificate)
```

Parameters

reqId – input
The identifier of the initialization request created by JNH_RA_preregister_nonpkix_user.

Certificate – output
A pointer to where the certificate (as an octetString) is written.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_register_nonpkix_user
- JNH_RA_nonpkix_user_check

Example

```
uint32 reqId;              // request identifier
octetString Certificate;   // pointer to where the certificate is stored
uint32 status;
status = JNH_RA_nonpkix_user_get_cert(reqId, &Certificate);
```

JNH_RA_preregister_crosscert

Creates a preregistration record for a subject CA. This record is used for cross-certification with another CA.

Syntax

```
uint32 JNH_RA_preregister_crosscert(const utf8String CAName,
                                    const utf8String UserName,
                                    uint32 ExpirationSeconds,
                                    const utf8String password,
                                    utf8String *PreRegistrationRecord)
```

Parameters

CAName – input
The name of the CA with which the subject CA is seeking to cross-certify. CAName a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

UserName – input
The name of the subject CA, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

ExpirationSeconds – input
The timeout length for the preregistration data.

password – input
The password to be used to encrypt the record.

PreRegistrationRecord – output
A record to be used by the subject CA to create a cross-certification request using JNH_preregister_crosscert.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_preregister_crosscert
- JNH_subject_submit_crosscert

Example

```
CAName = (utf8String) malloc(BUFSIZ);
IniReadString("General", "Issuer1", (char *)  CAName, BUFSIZ, NULL);
if ((status = JNH_RA_preregister_crosscert(CAName,
(utf8String)"/C=us/O=Iris Associates/OU=Subject CA", 0, (utf8String)"TEST",
&buf))) {
   return(status);
}
if ((fp = fopen("c:/ccertreg.reg", "w")) == NULL) {
   fprintf(stderr, "Cannot open out file\n");
   return(-1);
}
status = fwrite(buf, strlen((char *) buf), 1, fp);
fputc('\n', fp);
fclose(fp);
```

JNH_RA_preregister_nonpkix_user

Creates an initialization request (similar to the preregistration record) for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_preregister_nonpkix_user(const utf8String CAName,
                                       const utf8String UserName,
                                       uint32 * reqId)
```

Parameters

CAName – input
The Distinguished Name of the CA who will process the request. CAName is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

UserName – input
The Distinguished Name of the non-PKIX end entity. Username is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

reqId – output
A pointer to where the identifier of the initialization request is stored.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_register_nonpkix_user
- JNH_CA_preregister_nonpkix_user

Example

```
uint32 reqId;
uint32 status;
status = JNH_RA_preregister_nonpkix_user("/C=US:, "CN=A_USER", 0)
```

JNH_RA_preregister_user

Creates a preregistration record in preparation for a user registration.

Syntax

C++

```
uint32 JNH_RA_preregister_user(const utf8String CAName,
                               const utf8String UserName,
                               uint32 expirationSeconds,
                               const utf8String passwd,
                               utf8String * preregistrationRecord)
```

Java

```
int JAVA_RA_preregister_user(String CAName,
                             String UserName,
                             int expirationSeconds,
                             String passwd)
```

Parameters

CAName – input
The name of the CA to receive this request. CAName is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

UserName – input
Entity requesting the certificate. UserName is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

expirationSeconds – input
Timeout length for the pre-registration data.

Note: This parameter is not currently used.

passwd – input
Password to be used to encrypt the record.

preregistrationRecord – output
A record to be used by the user to register through JNH_register_user. This record contains key and transaction identifiers.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

A surrogate object will be created to preserve the security state associated with the preregistration record.

Related Functions

- JNH_preregister_user
- JNH_register_user

Example

C++

```
uint32 reqId;
uint32 status;
utf8String PreRegistrationRecord;
status = JNH_RA_preregister_user("/C=US", "CN=A_USER", 0,
         (utf8String) "rd_pw", &preregistrationRecord)
```

Java

```
int retVal;
retVal = JAVA_RA_preregister_user("/c=us", "cn=a_user", 0, "ra_pw");
if (retVal == 0)
{
   String PreRegistrationRecord = jonahInterface.retStr;
}
```

JNH_RA_register_nonpkix_user

Submits a registration request for a non-PKIX end entity.

Syntax

```
uint32 JNH_RA_register_nonpkix_user(const uint32 reqId)
```

Parameters

reqId – input
The identifier of the initialization request created by JNH_RA_preregister_nonpkix_user.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used for browser-based certificates.

Related Functions

- JNH_RA_preregister_nonpkix_user

Example

```
uint32 status;
uint32 reqId;     // request identifier
status = JNH_RA_register_nonpkix_user(reqId);
```

JNH_register_callbacks

Allows applications to react to events occurring on the PKIX servers, using two user-written API calls, JNH_Notify and JNH_Display.

Syntax

C++

```
uint32 JNH_register_callbacks
(void (* JNH_Notify)(uint32 eventId,
                     const utf8String name,
                     uint32 eventStatus),
 void (* JNH_Display)(uint32 displayType,
                      const utf8String message))
```

Java

```
int JAVA_register_callbacks()
```

Parameters

JNH_Notify parameters:

eventId – input
The identifier of the certificate request.

name – input
The name of the requestor.

eventStatus – input
The status of the certificate.

JNH_Display parameters:

displayType – input
The type of message being displayed.

message – input
The text of the message.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

All server GUIs must invoke JNH_register_callbacks at startup.

For more information about this API, see “Registering API callbacks” on page 10.

Example

C++

```
//display function
void display(uint32 type, const utf8String message)
{
      switch(type) {
      case DISPLAY_STATUSBAR:
        fprintf(stderr, "bar -> ");
        break;
      case DISPLAY_LOGERROR:
        fprintf(stderr, "log error -> ");
        break;
     case DISPLAY_LOGINFO:
        fprintf(stderr, "log info -> ");
        break;
      case DISPLAY_LOGDEBUG:
        fprintf(stderr, "log debug -> ");
        break;
     case DISPLAY_URGENTERROR:
        fprintf(stderr, "urg error -> ");
         break;
     case DISPLAY_URGENTINFO:
        fprintf(stderr, "urg info -> ");
        break;
     case DISPLAY_URGENTDEBUG:
        fprintf(stderr, "urg debug -> ");
        break;
      default:
        fprintf(stderr, "unk type -> ");
         break;
   }
      fprintf(stderr, "%s\n", message);
}

void notify(uint32 id, const utf8String name, uint32 status)
{
      uint32 foo;

      foo = status & 0xffff0000;
      if (foo == ObjStEECertReqActive)
{
          fprintf(stderr, "createCertReq: NAME=%s\tID=%d\tSTATUS=%x\n", name,
                  id, status);
   }

}

void main(int argc, char *argv[ ])
{
      int EeServer = 0;

      JNH_start_server(EeServer);
      JNH_register_callbacks(&notify, &display);

}
```

Java

```
int retVal;
retVal = jonahInterface.JAVA_register_callbacks( );
```

JNH_register_user

Submits a user registration request to the RA and processes the return reply.

Syntax

C++

```
uint32 JNH_register_user(uint32 reqId)
```

Java

```
int JAVA_register_user(int reqId)
```

Parameters

reqId – input
The identifier of the registration request.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The user information must have previously been established with a call to JNH_preregister_User.

Related Functions

- JNH_preregister_user

Example

C++

```
uint32 status;
uint32 reqId;     // id of registration request
status = JNH_register_user(reqId);
```

Java

```
int reqId;     // id of registration request
int retVal = jonahInterface.JAVA_register_user(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println("register user error= " + jonahInterface.retStr);
}
```

JNH_reject_registration

Rejects a registration request.

Syntax

C++

```
uint32 JNH_reject_registration(uint32 reqId,
                               utf8String message)
```

Java

```
int JAVA_reject_registration(int reqId, String message)
```

Parameters

reqId – input
The identifier of the registration request.

message – input
The reason for rejecting the registration request.

Usage

RA, CA

Return Values

0	Normal, successful completion
>0	An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_authorize_registration

Example

C++

```
uint32 status;
uint32 reqId;     // id of registration request
status = JNH_reject_registration(reqId, "Bad Key");
```

Java

```
// specify reason for rejection and select list item to reject
int reqId;     // id of registration request
int retVal;
String reason= "Bad Key";
retVal = jonahInterface.JAVA_reject_registration(reqId, reason);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println("RA Deny Certificate Error= "
                      + jonahInterface.retStr);
}
```

JNH_reject_revocation

Rejects a revocation request.

Syntax

C++

```
uint32 JNH_reject_revocation(uint32 reqId,
                             utf8String reason)
```

Java

```
int JAVA_reject_revocation(int reqId, String reason)
```

Parameters

reqId – input
The identifier of the revocation request being rejected.

reason – input
The reason the revocation request is rejected.

Usage

RA, CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

Certificates can be revoked for a number of reasons, including the following:

- The private key has been compromised.
- The CA has been compromised.
- The certificate holder’s affiliation has changed.
- The certificate is never used.

Related Functions

- JNH_authorize_revocation

Example

C++

```
uint32 rc;
uint32 reqId;     // id of revocation request to be rejected
rc = JNH_reject_revocation(reqId, (utf8String) "Found smart card");
```

Java

```
int reqId;     // id of revocation request to be revoked
int retVal = jonahInterface.JAVA_reject_revocation(reqId,
             "Found smart card");
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Reject Revocation error= "
                       + jonahInterface.retStr);
}
```

JNH_release_object

Unlocks an object in the object store without saving any changes.

Syntax

C++

```
uint32 JNH_release_object(uint32 reqId)
```

Java

```
int JAVA_release_object(int reqId)
```

Parameters

reqId – input
The identifier of the object

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

You must unlock objects using either this routine of JNH_save_object after you run inquire or set routines on them.

Related Functions

- JNH_reserve_object
- JNH_save_object

Example

C++

```
uint32 reqId;     // id of object
uint32 rc;
rc = JNH_release_object(reqId);
```

Java

```
int reqId;     // id of object
int retVal;
retVal = jonahInterface.JAVA_release_object(reqId);
```

JNH_release_octetString

Releases the memory associated with an octet string.

Syntax

```
uint32 JNH_release_octetString(octetString string)
```

Parameters

string – input
The string to be released.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_release_object

Example

```
octetString string;
status = JNH_release_octetString(serialNumber);
```

JNH_remove_certreq_extension

Removes an extension from a certificate request.

Syntax

```
uint32 JNH_remove_certreq_extension(uint32 xtype,
                                    uint32 reqId,
                                    const utf8String extID)
```

Parameters

xtype – input
The type of extension to be removed from the certificate request.

reqId – input
The identifier of the certificate request.

extID – input
The identifier of the extension to be removed from the certificate request. For private extensions, the identifier must be a numeric string in the dot format, such as 1.4.7.19.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_add_certreq_extension

Example

```
uint32 rc;
uint32 reqId;         // request identifier
uint32 xtype;         // extension type to be removed
utf8String extID;     // identifier of extension to be removed
status = JNH_remove_certreq_extension(reqId, xtype, extID);
```

JNH_request_CRL

Retrieves the CRL from the CA's BinBin.

Syntax

C++

```
uint32 JNH_request_CRL(utf8String issuer)
```

Java

```
int JAVA_request_CRL(String issuer)
```

Parameters

issuer – input
The name of the CRL issuer, which is a Distinguished Name in OSF syntax as described in "Parameter format" on page 7.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_create_CRL

Example

C++

```
uint32 rc;               // return code
utf8String issuer;       // CRL issuer
rc = JNH_create_CRL(issuer);
```

Java

```
int jonahInterface.JAVA_request_CRL(String issuer)
```

JNH_request_revocation

Submits a revocation request.

Syntax

C++

```
uint32 JNH_request_revocation(uint32 reqId)
```

Java

```
int JAVA_request_revocation(int reqId)
```

Parameters

reqId – output
The identifier of the revocation request being submitted.

Usage

EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_authorize_revocation
- JNH_revoke_certificate

Example

C++

```
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_request_revocation(reqId);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_request_revocation(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Request revocation error= "
                       + jonahInterface.retStr);
}
```

JNH_reserve_object

Reserves an object in the object store for inquire/set access.

Syntax

C++

```
uint32 JNH_reserve_object(uint32 reqid)
```

Java

```
int JAVA_reserve_object(reqId)
```

Parameters

reqId
The identifier of the object.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The object is read into memory, if necessary, and locked.

Objects must be locked with this routine before you can use inquire or set routines on them.

Related Functions

- JNH_save_object
- JNH_release_object

Example

C++

```
uint32 reqId;     // id of object
uint32 rc;
rc = JNH_reserve_object(reqId);
```

Java

```
int retVal;
int reqId;     // id of object
retVal = jonahInterface.JAVA_reserve_object(reqId);
```

JNH_revoke_certificate

Revokes the certificate.

Syntax

C++

```
uint32 JNH_revoke_certificate(uint32 reqId)
```

Java

```
int JAVA_revoke_certificate(int reqId)
```

Parameters

reqId – input
The identifier of the revocation request.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

A revocation record for this certificate will appear in the next CRL the CA creates with the specified key.

Related Functions

- JNH_create_CRL

Example

C++

```
uint32 rc;
uint32 reqId;     // id of the revocation request
rc = JNH_revoke_certificate(reqId);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_revoke_certificate(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval)
   System.out.println ("Revoke certificate error= "
                       + jonahInterface.retStr);
}
```

JNH_save_object

Updates the on-disk object store's record of the object to match the in-memory record and unlocks the object.

Syntax

```
uint32 JNH_save_object(uint32 reqId)
```

Parameters

reqId – input
The identifier of the object

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_reserve_object
- JNH_release_object

Example

```
uint32 rc;
uint32 reqId;     // id of object
rc = JNH_save_object(reqId);
```

JNH_server_login_pwd

Unlocks the server's credentials by directly supplying a user PIN.

Syntax

```
uint32 JNH_server_login_pwd(const utf8String pwd)
```

Parameters

pwd – input
The password to be used.

Usage

RA, CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This call is used during RA and CA initialization to log into the smart card.

Related Functions

- JNH_export_credential

Example

```
uint32 status;
utf8String passWord[] = "RApass";  //Password created when initializing RA

JNH_start_server(svrType_RA);
status = JNH_server_login_pwd(passWord);

return status;
```

JNH_Set_CCert_VerificationKey

Sets the subject CA's signing key for cross-certification.

Syntax

```
uint32 JNH_Set_CCert_VerificationKey(uint32 objId,
                                     utf8String serialNum)
```

Parameters

objId – input
The identifier of the cross-certification request created by JNH_preregister_crosscert.

serialNum – input
The serial number returned by JNH_CA_list_signing_key.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_preregister_crosscert
- JNH_CA_list_signing_key

Example

```
if (numKey > 0) {
   cout << "Setting the key info in certreq.." << endl;
   status = JNH_Set_CCert_VerificationKey(id, serialNum[0]);
   cout << "JNH_Set_CCert_VerificationKey returns " << status << endl;
   if (status) {
      return -1;
   }
}
```

JNH_set_certreq_basicConstraints

Sets the CA certificate indicator and the maximum path length values for a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_basicConstraints(uint32 reqId,
                                        IBOOL isCaCert,
                                        uint32 maxPathLen,
                                        IBOOL isMaxPathLenUnlimited)
```

Java

```
int JAVA_set_certreq_basicConstraints(int reqId,
                                      boolean isCaCert,
                                      int maxPathLen)
```

Parameters

reqId – input
The identifier of the certificate request.

isCaCert – input
Sets the CA certificate indicator. Set to true if this is a CA certificate.

maxPathLen – input
Sets the maximum path length allowed. A maxPathLen value of –1 means the path length is unconstrained.

isMaxPathLenUnlimited – input
Specifies whether or not the maximum path length is unlimited.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

If isCaCert is false (the certificate is an EE certificate), this routine removes any basic constraints extension from the certificate request.

Related Functions

- JNH_inquire_certreq_basicConstraints

Example

C++

```
uint32 rc = 0
uint32 reqId;     // id of certificate request
boolean isCaCert = TRUE;
uint32 maxPathLen = 27;
rc = JNH_set_certreq_basicConstraints(reqId, isCaCert, maxPathLen);
```

Java

```
int reqId;     // id of certificate request
int maxPathLen = 27;
boolean isCaCert = TRUE;
int retVal = jonahInterface.JAVA_set_certreq_basicConstraints(reqId,
              isCaCert, maxPathLen);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("SetCertBasicConstraints failed, = "
                    + jonahInterface.retStr);
}
```

JNH_set_certreq_endDate

Sets the expiration date in a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_endDate(uint32 reqId,
                               utcDateTime notAfter)
```

Java

```
int JAVA_set_certreq_endDate(int reqId,
                             int year,
                             int month,
                             int day)
```

Parameters

reqId – input
The identifier of the certificate request.

notAfter – input
The expiration date for the certificate request.

year – input
The year of the expiration date.

month – input
The month of the expiration date.

day – input
The day of the expiration date.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_endDate

Example

C++

```
uint32 reqId;     // id of certificate request
uint32 rc;
utcDateTime notAfter;
notAfter.year = 2001;
notAfter.month = 11;
notAfter.day = 10;
notAfter.hour = 9;
notAfter.min = 8;
notAfter.sec = 7;
rc = JNH_set_certreq_endDate(reqId, notAfter);
```

Java

```
int reqId;     // id of certificate request
int yr = 1999;
int month = 5;
int day = 12;
int retVal = jonahInterface.JAVA_set_certreq_endDate(reqId, yr,
             month, day);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("SetEndDate Error= " + jonahInterface.retStr);
}
```

JNH_set_certreq_issuer

Sets the issuer name in a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_issuer(uint32 reqId,
                              const utf8String issuer)
```

Java

```
int JAVA_set_certreq_issuer(int reqId,
                            String issuer)
```

Parameters

reqId – input
The identifier of the certificate request.

issuer – input
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_issuer

Example

C++

```
uint32 reqId;     // id of certificate request
uint32 rc;
utf8String SISSUER= "/C=UK";
rc = JNH_set_certreq_issuer(reqId, SISSUER)
```

Java

```
String issuerName= "/C=UK";
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_set_certreq_issuer(reqId, issuer);
if (retVal != 0)
{
    System.out.println ("SetIssuer Error= " + jonahInterface.retStr);
}
```

JNH_set_certreq_keyUsage

Sets the key usage in a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_keyUsage(uint32 reqId,
                                keyUsage_t keyUsage)
```

Java

```
int JAVA_set_certreq_keyUsage(int reqId, int KeyUsage)
```

Parameters

reqId – input
The identifier of the certificate request.

keyUsage – input
The sum of the numeric values (from the table below) for each usage flag being requested.

Extension	**Usage**
1	Digital Signature
2	Non-repudiation
4	Key Encipherment
8	Data Encipherment
16	Key Agreement
32	Key Cert Sign
64	CRL Sign
128	Encipher Only
256	Decipher Only

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_keyUsage

Example

C++

```
uint32 rc;
uint32 reqId;     // id of certificate request
KeyUsage_t KeyUsage;
```

```
KeyUsage = USAGE_digitalSignature |
           USAGE_nonRepudiation;
rc = JNH_set_certreq_keyUsage(reqId, keyUsage);
```

Java

```
int keyUsageVal = jonahInterface.usageDigitalSignature |
                  jonahInterface.usageNonRepudiation;
int reqId;     // id of certificate request
int retVal = jonahInterface.JAVA_set_certreq_keyUsage(reqId, keyUsageVal);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Set Certificate Request Key Usage Error= "
                       + jonahInterface.retstr);
}
```

JNH_set_certreq_privkey_EE

Sets the key length and algorithm for the EE-generated private key for the certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_privkey_EE(uint32 reqId,
                                  const utf8String algorithm,
                                  uint32 keylength)
```

Java

```
int JAVA_set_certreq_privkey_EE(int reqId,
                                String algorithm,
                                int keyLength)
```

Parameters

reqId – input
The identifier of the certificate request.

algorithm – input
The algorithm to use (such as RSA or DSA).

keyLength – input
The keyLength for the private key in bits. The choices for this length vary depending on the algorithm used to create the certificate.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_privkey_EE

Example

C++

```
uint32 rc;
uint32 reqId;     // id of certificate request
utf8String alg= "id_dsa";
uint32 keyLength = 512;
rc = JNH_set_certreq_privkey_EE(reqId, alg, keylength);
```

Java

```
int retVal;
int keyLength = 512
String alg= "id_dsa";
int reqId;     // id of certificate request
retVal = jonahInterface.JAVA_set_certreq_privkey_EE(reqId, algorithm, keyLength);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.println.out("Set Certificate Request Private Key Error= "
                     + jonahInterface.retStr);
}
```

JNH_set_certreq_startDate

Sets the starting date in a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_startDate(uint32 reqId,
                                 utcDateTime notBefore)
```

Java

```
int JAVA_set_certreq_startDate(int reqId, int year, int month, int day)
```

Parameters

reqId – input
The identifier of the certificate request.

notBefore – input
The starting date for the certificate request.

year – input
The year of the starting date.

month – input
The month of the starting date.

day – input
The day of the starting date.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_startdate

Example

C++

```
uint32 rc;
uint32 reqId;     // id of certificate request
utcDateTime notBefore;
notBefore.year = 2002;
notBefore.month = 5;
notBefore.day = 12;
notBefore.hr = 8;
notBefore.min = 7;
notBefore.sec = 6;
rc = JNH_set_certreq_startDate(reqId, notBefore);
```

Java

```
int reqId;     // id of certificate request
int retVal;
retVal = jonahInterface.JAVA_set_certreq_startDate(reqId, 1999, 5, 12);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("StartDate error= " + jonahInterface.retStr);
}
```

JNH_set_certreq_subject

Sets the subject name in a certificate request.

Syntax

C++

```
uint32 JNH_set_certreq_subject(uint32 reqId,
                               const utf8String subject)
```

Java

```
int JAVA_set_certreq_subject(int reqId, String subject)
```

Parameters

reqId – input
The identifier of the certificate request.

subject – input
The subject name of the certificate request, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

Usage

RA, CA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_certreq_subject

Example

C++

```
uint32 rc;
uint32 reqId;     // id of certificate request
rc = JNH_set_certreq_subject(reqId, (utf8String)
     "/OU=Company /CN=S_user");
```

Java

```
int retVal;
int reqId;     // id of certificate request
retVal = jonahInterface.JAVA_set_certreq_subject(reqId,
         "/OU=Company /CN=S_user);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Error in set_certreq_subject= "
                      + jonahInterface.retStr);
}
```

JNH_set_certreq_subjectKeyInfo

Copies the subject public key and algorithm from a specified self-signed certificate into a certificate request.

Syntax

```
uint32 JNH_set_certreq_subjectKeyInfo(uint32 reqId,
                                      const octetString serialNumber)
```

Parameters

reqId – input
The identifier of the certificate request.

serialNumber – input
The serial number of the self-signed certificate. This number is created by a call to JNH_get_self_subjectKeyInfo.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_get_self_subjectKeyInfo

Example

```
uint32 status;
uint32 reqId;
octetString serialNum;
/* Using value from JNH_get_self_subjectKeyInfo assuming first value
/* is self-signed certificate
/*
status = JNH_set_certreq_subjectKeyInfo(reqId, serialNum[0]);
```

JNH_set_IniLdapAuthName

Sets the LDAP AuthName information for bootstrap in the .ini file.

Syntax

```
uint32 JNH_set_IniLdapAuthName(const utf8String authname)
```

Parameters

authname – input
The LDAP AuthName to set in the .ini file.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniLdapAuthName
- JNH_set_IniLdapServer

Example

```
char authname[100];
cout << "enter ldap authname" << endl;
cin.getline(authname, sizeof(authname));
status = JNH_set_IniLdapAuthName(objid, (utf8String)authname);
```

JNH_set_IniLdapAuthPwd

Sets the LDAP server password information in the .ini file.

Syntax

```
uint32 JNH_set_IniLdapAuthPwd(const utf8String pwd)
```

Parameters

pwd – input
The LDAP server password to set in the .ini file.

Usage

CA. RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniLdapAuthPwd
- JNH_set_IniLdapAuthName

Example

```
char password[100];
cout << "enter ldap password" << endl;
cin.getline(password, sizeof(password));
status = JNH_set_MyName(objid, (utf8String)password);
```

JNH_set_IniLdapServer

Sets the LDAP server name for bootstrap in the .ini file.

Syntax

```
uint32 JNH_set_IniLdapServer(const utf8String server)
```

Parameters

server – input
The name of the LDAP server.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniLdapServer
- JNH_set_IniTcpPort

Example

```
char server[100];
cout << "enter ldap server" << endl;
cin.getline(server, sizeof(server));
status = JNH_set_MyName(objid, (utf8String)server);
```

JNH_set_IniTcpHost

Sets the TCP host information for bootstrap in the .ini file.

Syntax

```
uint32 JNH_set_IniTcpHost(const utf8String host)
```

Parameters

host – input
The name of the TCP host.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniTcpHost
- JNH_set_MyName

Example

```
char host[100];
cout << "enter my host" << endl;
cin.getline(host, sizeof(host));
status = JNH_set_MyName(objid, (utf8String)host);
```

JNH_set_IniTcpPort

Sets the TCP port information for bootstrap in the .ini file.

Syntax

```
uint32 JNH_set_IniTcpPort(const utf8String port)
```

Parameters

port – input
The TCP port information.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniTcpPort
- JNH_set_IniTcpHost
- JNH_set_MyName

Example

```
char port[100];
cout << "enter my port" << endl;
cin.getline(port, sizeof(port));
status = JNH_set_MyName(objid, (utf8String)port);
```

JNH_set_MyName

Sets the MyName field in the .ini file, as well as in the certificate for bootstrap.

Syntax

```
uint32 JNH_set_MyName(uint32 reqId,
                      const utf8String name)
```

Parameters

reqId – input
The identifier of the bootstrap request created by JNH_create_BootStrap.

name – input
The name to be set in the MyName field.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The information from this call is not added to the .ini file until after JNH_BootStrap is successfully called.

Related Functions

- JNH_get_IniMyName
- JNH_BootStrap

Example

```
char myname[100];
cout << "enter my name " << endl;
cin.getline(myname, sizeof(myname));
status = JNH_set_MyName(objid, (utf8String)myname);
```

JNH_set_RA_URL

Sets the RA’s URL, subject, and password in a revocation request.

Syntax

C++

```
uint32 JNH_set_RA_URL(uint32 reqId,
                          const utf8String url,
                          const utf8String subject,
                          const utf8String passwd)
```

Java

```
int JAVA_set_RA_URL(int reqId,
                    String url,
                    String subject,
                    String passwd)
```

Parameters

reqId – input
The identifier of the revocation request.

url – input
The RA’s URL.

subject – input
The name of the subject, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

passwd – input
The smart card’s user password.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

The URL is not stored in a certificate, so it cannot be added to the revocation request from the certificate. The URL must be obtained by some other means (such as, prompting for it). The typical scenario is to call:

1. JNH_create_revreq_from_certificate to create a revocation request,
2. JNH_set_RA_URL to set the RA’s URL and other required information into the request,
3. JNH_request_revocation to send the revocation request to the RA.

Related Functions

- JNH_create_revreq_from_certificate
- JNH_request_revocation

Example

C++

```
uint32 retVal;
  uint32 reqId;                                     // rev req ID
  utf8String url = "pkix://localhost:829";          // RA's URL
  utf8String subject = "/C=US/O=IBM/OU=Test/CN=One";// subject DN
  utf8String passwd = "passwd";                     // password

  retVal = JNH_set_RA_URL(reqId,
                          url,
                          subject,
                          passwd);
```

Java

```
int retVal;
int reqId = 3;                                  // rev req ID
String url = "pkix://localhost:829";            // RA's URL
String subject = "/C=US/O=IBM/OU=Test/CN=One";  // subject DN
String passwd = "passwd";                       // password

retVal = jonahInterface.JAVA_set_RA_URL(reqId,
                                        url,
                                        subject,
                                        passwd);
```

JNH_set_revreq_certIssuer

Sets the name of the certificate issuer in the CertDetails section of one of the revocations in the revocation request.

Syntax

C++

```
uint32 JNH_set_revreq_certIssuer(uint32 reqId,
                                 uint32 index,
                                 const utf8String certIssuer)
```

Java

```
int JAVA_set_revreq_certIssuer(int reqId, String certIssuer)
```

Parameters

reqId – input
The identifier of the revocation request being queried. The identifier must be of type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

certIssuer – input
The name of the certificate issuer, which is a Distinguished Name in OSF syntax as described in “Parameter format” on page 7.

This parameter must not be null, nor a zero-length string.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_revreq_certIssuer

Example

C++

```
uint32 reqId;     // id of revocation request
char issuername2[]= "/C=US/O=Iris /OU=Test/CN=CSPTest2";
utf8String issuer = (unsigned char*) &issuername2;
uint32 rc;
rc = JNH_set_revreq_certIssuer(reqId, (uint32)0, issuer);
```

Java

```
int retVal;
int reqId;    // id of revocation request
retVal = jonahInterface.JAVA_set_revreq_certIssuer(reqId,
         "/C=US/O=Iris/OU=Test/CN=Test2");
if (retVal != 0)
```

```
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Set Certificate Issuer error= "
                       + jonahInterface.retStr;
}
```

JNH_set_revreq_certserialnumber

Sets the certificate serial number for one of the revocations in the revocations request.

Syntax

C++

```
uint32 JNH_set_revreq_certserialnumber(uint32 reqId,
                                       uint32 index,
                                       octetString * serialNumber)
```

Java

```
int JAVA_set_revreq_certserialnumber(int reqId,
                                     String serialNumber)
```

Parameters

reqId – input
The identifier of the revocation request being queried. The identifier must be of type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

serialNumber – input
The serial number of the certificate whose revocation is being requested, in standard ASN.1 binary format for integers.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_revreq_certserialnumber
- JNH_inquire_revreq_certserialnumbers

Example

C++

```
#define test4serialnum 5432
Serialnum = test4serialnum;
SerialNumber.data = (unsigned char*) &Serialnum;
SerialNumber.length = sizeof(Serialnum);
uint32 reqId;     // id of revocation request
uint32 rc;
rc = JNH_set_revreq_certserialnumber(reqId, (uint32)0, &serialNumber);
```

Java

```
int retVal;
int reqId;     // id of revocation request
retVal = jonahInterface.JAVA_set_revreq_certserialnumber(reqId, "5432");
```

```
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval)
   System.out.println ("Set CertSerialNumber error= "
                       + jonahInterface.retStr);
}
```

JNH_set_revreq_hold_instruction_code

Sets the hold_instruction_code for a certificate being revoked in a revocation request.

Syntax

```
uint32 JNH_set_revreq_hold_instruction_code(uint32 reqId,
                                             uint32 index,
                                             const octetString code)
```

Parameters

reqId – input
The identifier of the revocation request being queried. The identifier must be of type ObjClTypeRev.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

code – input
The object ID of the hold instruction code, in the standard ASN.1 binary format for object identifiers.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API should only be used for revocations whose reason is CERTIFICATE_HOLD.

Related Functions

- JNH_inquire_revreq_hold_instruction_code

Example

```
char inst1[]= "instruction1";
octetString inHold;
inHold.data = (unsigned char *) &inst1;
inHold.length = strlen(inst1);
int rc;
int reqId;     // id of revocation request
rc = JNH_set_revreq_hold_instruction_code(reqId, (uint32)0,
     &inHold);
```

JNH_set_revreq_invalidityDate

Sets the invalidity date in a certificate being revoked by a revocation request.

Syntax

C++

```
uint32 JNH_set_revreq_invalidityDate(uint32 reqId,
                                     uint32 index,
                                     utcDateTime InvalidityDate)
```

Java

```
int JAVA_set_revreq_invalidityDate(int reqId,
                                   int year,
                                   int month,
                                   int day)
```

Parameters

reqId – input
The identifier of the revocation request being queried. The identifier must be type ObjClTypeRev.

Index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

InvalidityDate – input
A utcDate time structure containing the invalidity date.

year – input
The year of the invalidity date.

month – input
The month of the invalidity date.

day – input
The day of the invalidity date.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_revreq_invalidityDate

Example

C++

```
utcDateTime inTime;
inTime.year = 2000;
inTime.month = 5;
inTime.day = 13;
inTime.min = 0;
inTime.sec = 0;
inTime.msec = 0;
int reqId;     // id of revocation request
int rc;
rc = JNH_set_revreq_invalidityDate(reqId, (uint32)0, inTime);
```

Java

```
int retVal;
int reqId;     // id of revocation request
int year = 1999;
int month = 5;
int day = 13;
retVal = jonahInterface.JAVA_set_invalidityDate(reqId, year, month,day);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Set Invalidity Date error= "
                       + jonahInterface.retStr);
}
```

JNH_set_revreq_reason

Sets the reason code for a certificate being revoked in a revocation request.

Syntax

C++

```
uint32 JNH_set_revreq_reason(uint32 reqId,
                             uint32 index,
                             int * reasonFlags)
```

Java

```
int JAVA_set_revreq_reason(int reqId,
                           int reason)
```

Parameters

reqId – input
The identifier of the revocation request being queried.

index – input
The zero-based index of the certificate, within the list of certificates being revoked by this request, from which the caller wants to extract information.

reasonFlags – input
The reason code to be set.

The reason codes are as follows:

0	REV_REASON_NONE
1	REV_REASON_UNUSED
2	REV_REASON_KEY_COMPROMISE
4	REV_REASON_CA_COMPROMISE
8	REV_REASON_AFFILIATION_CHANGED
16	REV_REASON_SUPERSEDED
32	REV_REASON_CESSATION_OF_OPERATION
64	REV_REASON_CERTIFICATE_HOLD

For a complete list of the revocation reason codes and their definitions, see Include\ASN1\X509.h REASON_*.

Notes:

1. This parameter must not be null.
2. The reasonFlags are not modified by calling JNH_set_revreq_reason and can be reused or freed after the call.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_inquire_revreq_reason

Example

C++

```
int inReasons = REV_REASON_KEY_COMPROMISE;
uint32 rc;
uint32 reqId;     // id of revocation request
rc = JNH_set_revocation_request(reqId, (uint32)0, &inReasons);
```

Java

```
int reqId;     // id of revocation request
int retVal;
int reasonBinary = jonahInterface.REV_REASON_KEY_COMPROMISE;
retVal = jonahInterface.JAVA_set_revreq_reason(reqId, reasonBinary);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println ("Set revreq reason error= "
                       + jonahInterface.retStr);
}
```

JNH_set_server_location

Sets the background server location and checks that the server is running in that location.

Syntax

```
uint32 JNH_set_server_location(utf8String address,
                               uint32 adminPort)
```

Parameters

address – input
The TCP/IP name of the CA or RA background server machine.

adminPort – input
The administrative port number for the port on which the server is listening.

Usage

CA, RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_shutdown_UI

Example

```
utf8String server;     // TCP/IP name of server machine
uint32 adminPort;      // Admin port number server is listening to
uint32 status;

status = JNH_set_server_location(server, adminPort);
```

JNH_shutdown_UI

Allows the server to clean up resources associated with a background server user interface session.

Syntax

```
uint32 JNH_shutdown_UI()
```

Parameters

none

Usage

RA, CA, EE

Return Values

REMJNH_OK
Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_initialize_UI

Example

```
JNH_shutdown_UI();
```

JNH_start_server

Initializes a server and opens or creates an object store.

Syntax

C++

```
uint32 JNH_start_server(int serverType,
                        char * inifile = "")
```

Java

```
int JAVA_start_server(int serverType, String inifile)
```

Parameters

serverType – input
Specifies which server is being started:

EE	Init_EE 0
RA	Init_RA 1
CA	Init_CA 2
RA client	Init_RC 3
CA client	Init_CC 4

inifile – input
The fully-qualified name of the server's .ini file. If omitted, the default .ini file name for that entity is assumed.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Example

C++

```
JNH_start_server(0);
```

Java

```
jonahInterface.JAVA_start_server(0);
```

JNH_stop_server

Shuts down a server in an orderly fashion.

Syntax

C++

```
uint32 JNH_stop_server(int serverType)
```

Java

```
int JAVA_stop_server (int serverType)
```

Parameters

serverType – input
The type of server to be shut down. Specify one of the following:

init_EE
End entity.

init_RA
Registration authority.

init_CA
Certificate authority.

Usage

EE, RA, CA

Return Values

0 Normal, successful completion

>0 An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_start_server

Example

C++

```
uint32 status=0;

uint32 JNH_stop_server(serverType);
```

Java

```
uint32 status=0;

status = JAVA_stop_server(serverType);
```

JNH_store_RA_URL

Extracts the RA’s URL from the object store and stores it in the RenewRevokeCertList section of the .ini file. The URL is stored in the following format:

serialNumber:IssuerDn=RA’s URL

Syntax

C++

```
uint32 JNH_store_RA_URL(uint32 reqId)
```

Java

```
int JAVA_store_RA_URL(int reqId)
```

Parameters

reqId – input
The identifier of the request in the object store from which the RA URL is being extracted.

Usage

CA, RA, EE

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Remarks

This API is used to maintain a list of the issuing RAs of certificates on a smart card so that a renewal or revocation request can be sent to the same RA. The API is called after a certificate has been issued and is being returned to the EE to export to a smart card.

Example

C++

```
uint32 retVal;
uint32 reqId;
retVal = JNH_store_RA_URL(reqId);
```

Java

```
int retVal = JAVA_store_RA_URL(reqId);
```

JNH_subject_submit_crosscert

Submits a subject CA's request for cross-certification.

Syntax

```
uint32 JNH_subject_submit_crosscert(uint32 objId)
```

Parameters

objId – input
The identifier of the cross-certification request created by JNH_preregister_crosscert.

Usage

CA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_preregister_crosscert

Example

```
status = JNH_subject_submit_crosscert(id);
cout << "JNH_subject_submit_crosscert returns " << status << endl;
```

JNH_validate_registration

Validates a certificate registration request for authenticity, ensures that the version equals 2, the signing algorithm is present, and that the incoming certificate policy values are allowed to be set by the caller. Also sets the algorithm name from the .ini file.

Syntax

C++

```
uint32 JNH_validate_registration(uint32 reqId)
```

Java

```
int JAVA_validate_registration(int reqId)
```

Parameters

reqId – input
The identifier of the registration request.

Usage

RA

Return Values

0 Normal, successful completion

> 0
An error occurred. See the apimsg.h file for details.

Related Functions

- JNH_authorize_registration
- JNH_reject_registration

Example

C++

```
uint32 reqId;     // id of registration request
uint32 status;
status = JNH_validate_registration(reqId);
```

Java

```
int reqId;     // id of registration request
int retVal = jonahInterface.JAVA_validate_registration(reqId);
if (retVal != 0)
{
   jonahInterface.JAVA_get_error(retval);
   System.out.println("RA Validate Certificate Error= "
                      + jonahInterface.retStr);
}
```

Appendix A. Using the ASN.1 class library

PKIX constructs use Abstract Syntax Notation One (ASN.1), an industry standard for the formal notation of abstract data types and structures. ASN.1 is a data definition language that uses simple syntax and lexical rules with a small set of keywords and is recursively defined.

This chapter briefly describes how ASN.1 is used in PKIX and presents some code samples to help you understand its function. More information on ASN.1, in a number of viewable formats, can be found at http://www.rsa.com/rsalabs/pubs/PKCS/.

ASN.1 data types

ASN.1 notation provides the following base data types listed below. PKIX uses these base ASN.1 data types to build a set of its own data types, which are listed separately.

ANY	An arbitrary value of an arbitrary type.
CHOICE	A union of one or more alternatives.
INTEGER	An arbitrary integer whose value can be positive, negative, zero, or any magnitude.
BOOLEAN	A boolean value.
NULL	Null value.
OBJECT IDENTIFIER	An object identifier, sequence of integer components that identify an object, attribute type, or registration authority identifying other object identifiers.
BIT STRING	An arbitrary string of bits (ones and zeroes).
OCTET STRING	A string of octets (eight values).
PrintableString	An arbitrary string of printable characters.
T61String	A arbitrary string of T.61 characters. T.61 is an 8-bit extension to the ASCII character set.
IA5String	An arbitrary string of ASCII characters.
SEQUENCE	An ordered collection of one or more types.
SEQUENCE OF	An ordered collection of zero or more occurrences of a given type.
SET	An unordered collection of one or more types.
SET OF	An unordered collection of zero or more occurrences of a given type.
UTCTime	A coordinated universal time or Greenwich Mean Time (GMT) value.

The following example shows how an X.509 certificate is defined in ASN.1 notation:

```
Certificate ::= SEQUENCE {
     tbsCertificate      TBSCertificate,
     signatureAlgorithm  AlgorithIdentifier,
     signatureValue      BIT STRING }
```

```
TBSCertificate ::= SEQUENCE {
     version                [0] EXPLICIT Version Default v1
     serialNumber           CertificateSerialNumber,
     signature              AlgorithmIdentifier,
     issuer                 Name
     validity               Validity,
     subject                Name
     subjectPublickKeyInfo  SubjectPublicKeyInfo,
     issuerUniqueID         [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                                  --If present, version shall be v2 or v3
     subjectUniqID          [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                                  --If present, version shall be v2 or v3
     extensions            [3]  EXPLICIT Extensions OPTIONAL,
                                  --If present, version shall be v3
}
Version ::= INTEGER { v1(0), v2(1), v3(2)
```

PKIX interface data types

The following data types, built from the ASN.1 data types defined above, are used in PKIX:

uint32	A 32-bit integer.
utf8String	A UTF-8 encoded, null-terminated string.
octetString	An array of bytes.
IBOOL	A boolean value.
utcDateTime	A date, represented as an octetstring containing the value part of the BER encoding of a GeneralizedTime value representing the date.
keyUsage_t	A set of bits that represent the possible functions for which a key can be used.

Bits	**Key Function**
0	DigitalSignature
1	NonRepudiation
2	KeyEncipherment
3	DataEncipherment
4	KeyAgreement
5	KeyCertSign
6	CRLSign
7	EncipherOnly
8	DecipherOnly

BER and DER encoding

ASN.1 notation provides rules for encoding values. The Basic Encoding Rules (BER) enable you to represent any ASN.1 type and the value associated with it as an octet string. A subset of BER encoding is the Distinguished Encoding Rules (DER). DER encoding allows only one method for representing an ASN.1 value and is used when you require unique encoding of an octet string, such as a computed digital signature.

ASN.1 class library hierarchy

Illustrated below is the ASN.1 class library hierarchy. The indentation represents the inheritance relationship.

```
ans_object
   asn_primitive
      asn_null
      asn_integer
      asn_bitstring
         asn_namedbits
      asn_boolean
      asn_octetstring
         asn_charstring
            asn_ia5string
            asn_directorystring
            asn_UTF8string
            asn_teletexstring
            asn_printablestring
   asn_composite
      asn_sequence
      asn_sorted
         asn_set
      asn_choice
   asn_any
```

Abstract classes

ASN.1 uses two abstract classes: asn_object and asn_composite. These classes are abstract because they define pure virtual functions.

```
 class      asn_object      { encode_value(buffer_t & buf) const=0;
                             decode_value(r_buffer_t & buf,
                                          uint32 value_length)=0; ....;};
 class      asn_composite    { ... };
```

These virtual methods are implemented at the lower level of ASN.1 classes corresponding to concrete ASN.1 data types.

The following code segments illustrate manipulating ASN.1 objects:

```
class x509_Extension : public asn_sequence {
  asn_oid extrnID;
  asn_boolean critical;
  asn_octetstring extnValue;..... // other definitions
};
class x509_Extensions : public
  asn_sequenceof<x509_Extension> {...};


CertReqMsg certReqMsg= ...
x509_Extensions * pextensions;
pextensions=&(certreqmsg->certReq.certTemplate.extensions);

x509_Extension * this_ext;          // delete an extension
asn_oid undesired_oid; ....;
uint32 num_ext= pextensions->get_child_count();
for (uint32 ext_index = 0; ext_index < num_ext; ext_index++)
{
    this_ext = pextensions->get_child(ext_index);
    if (undesired_oid == this_ext->extnID) {
       status = pextensions->delete_child (ext_index);
       break; }}
                                    // add a new extension
this_ext = pextensions->add_child(); // add a new member to the
```

```
                        // sequence of extensions

                                        // set named oid
status = this_ext->extnID.set_value("testId",strlen("testId"));

buffer_t buf;
status = buff.clear();
status = my_octets.write(buf);


asn_octetstring my_octets;
status = my_octets.set_value("test"),strlen("test"));

status = this_ext->extnValue.set_value(buf.data,
    buf.data_len);
status = this_ext->critical.set_value(false);
```

The following example shows how to ″flatten″ an ASN.1 value:

```
buffer_t this_buff;
this_buf.clear();
status = this_ext->extnID.write(this_buff);
status = this_ext->extnValue.write(this_buff);
status = this_ext->critical.write(this_buff);

this_buf.clear();
                                        // or simply to flatten the whole extension object
status = this_ext->write(this_buff);
```

Notes:

1. Use the read method to reconstruct an asn_object given its encoded buffer.
2. Use the write method to encode or flatten an ASN.1 object.

ASN.1 header files

To access the basic ASN.1 capabilities, you must include in your application (using #include statements) the asnbase.h header file. In addition, the asnstrng.h header file provides the support for various character string types including codeset conversions for the following:

- PrintableString
- Teletex/T.61
- IA5
- 16- and 32-bit encodings of Unicode

The asnnames.h header file defines a class representing X.500 names and supports both the X.500 printable syntax (/C=us/O=. . .) and the RFC 1779 syntax (CN=Myname; OU=Security; O=. . .).

Appendix B. PKIX programming model

This section describes the IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms (PKIX) programming model. Information in this section includes:

- An introduction to the IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms and its components
- Programming the certificate authority (CA) and registration authority (RA) bootstraps
- Sample bootstrap C++ sequence calls
- Enrolling the RA with the CA
- Sample RA enrollment code
- Programming to the end entity (EE)
- Sample EE certificate life cycle code
- Programming to the RA
- Programming to the CA

Introduction

The IBM SecureWay X.509 Public Key Infrastructure for Multiplatforms (PKIX) certificates consist of the following interactive components:

- End entity (EE)
- Registration authority (RA)
- Certificate authority (CA)

The EE component presents an interface to a programming entity or to a human entity to initiate certificate life cycle requests targeted to the RA, such as initiating a certificate request or a revocation request. The EE also generates the private and public key pairs for a prospective certificate holder.

The RA supports administrative functions, such as the registration of subjects (prospective holders of X.509 certificates), in preparation for subjects to issue certificate requests. The RA also validates that a subject is entitled to have the attributes being requested in a certificate, and the RA securely verifies the authenticity of a certificate request being appropriately associated with a previous registration record for the subject.

The CA is the PKIX trusted authority for issuing and certifying X.509 certificates.

The interactions among the PKIX components as illustrated in Figure 1 on page 210 start at the EE, extend to the RA, and then move to the CA. Replies are propagated back from the CA to the RA and then to the EE.

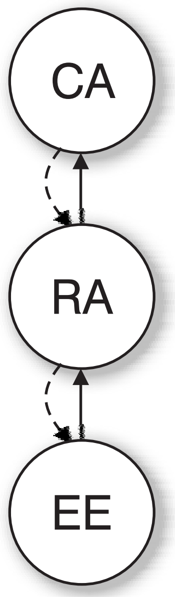

Figure 1. Hierarchical relationship among the PKIX components

The functions of these three components, as exposed through their respective human graphic interfaces, are also exposed to application developers through a set of application programming interfaces (APIs). The object store associated with each of the PKIX components is central to these APIs. However, the object store remains transparent, so programmers will need to refer to object store elements using only the numeric identifiers. Each of these identifiers is assigned when the object is first created. The latter step is also exposed to programming through its own interface.

Figure 2 on page 211 illustrates the main programming steps that a PKIX programmer needs to adopt. They include the following:

1. Performing a manual initialization step by running the `initsc` program for the three components EE, RA, and CA, whereby a respective virtual smart card key store is initialized.
2. Bootstrapping the CA and the RA.
3. Enrolling the RA with the CA.
4. Programming the certificate life cycle and manipulation with the EE, RA, and CA.

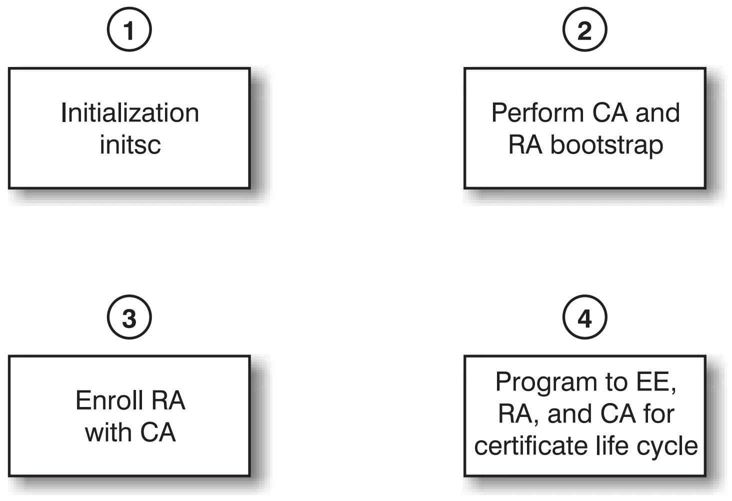

Figure 2. Overall PKIX programming steps

Programming the CA and RA bootstraps

The outcome of the bootstrap programming steps is the generation of a self-signed certificate for the CA (also known as a certificate for the root CA) and a certificate for the RA. The CA certificate may get posted to the LDAP directory server when applicable and desirable. Additionally, these certificates are stored in the object store attached to the CA and to the RA respectively, while associated private keys are stored in their respective virtual smart cards. Figure 3 on page 212 illustrates the overall CA and RA bootstrap programming sequence.

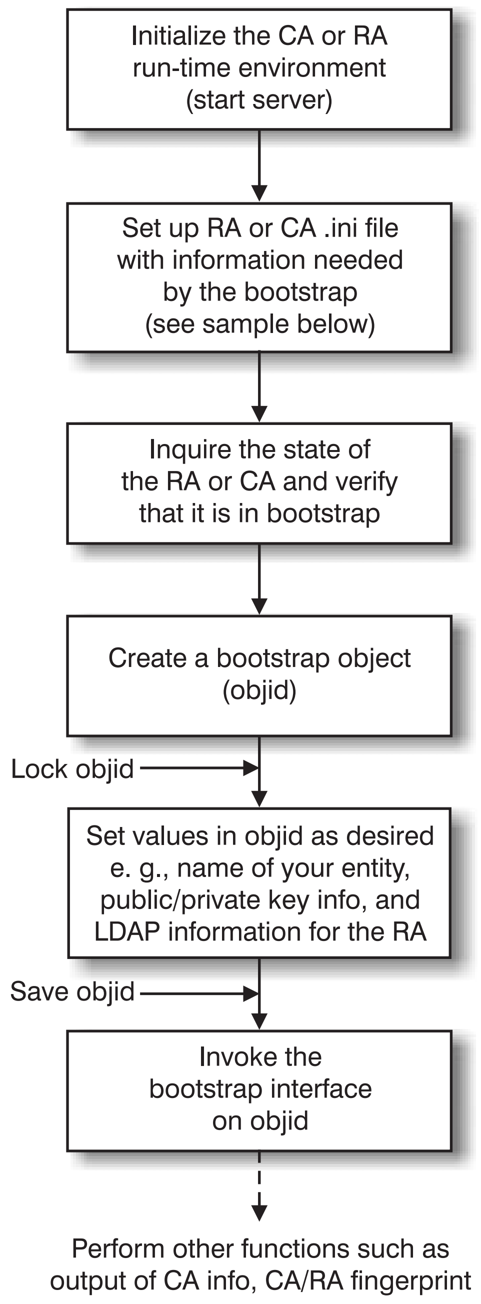
Initialize the CA or RA run-time environment (start server)
Set up RA or CA .ini file with information needed by the bootstrap (see sample below)
Inquire the state of the RA or CA and verify that it is in bootstrap
Create a bootstrap object (objid)
Lock objid
Set values in objid as desired e. g., name of your entity, public/private key info, and LDAP information for the RA
Save objid
Invoke the bootstrap interface on objid
Perform other functions such as output of CA info, CA/RA fingerprint

Figure 3. Overall CA and RA bootstrap programming steps

Sample bootstrap C++ sequence calls

Note: Calls to JNH APIs below will need to be converted into corresponding C++ invocations.

```
int server_type = svrType_CA; // or svrType_RA
uint32 status;
if (status = JNH_start_server(server_type),(char*) "c:\pkix\myRA.ini") return status;
if (serverstatus != svrSt_Bootstrap) return serverstatus;


// Set up .ini file bootstrap information.
if (status = JNH_set_IniTcpHost((utf8String)"9.53.91.93")) return status;
if (status = JNH_set_IniTcpPort((utf8String)"1829") return status;

   // The following sets up LDAP authentication information that
   // the RA uses when connecting to a configured LDAP server.

if (server_type == svrType_RA) { // RA only

  status = JNH_set_IniLdapServer((utf8String)"myLDAPserver.mydomain.com:371");
  if (status) return status;

  status = JNH_set_IniLdapAuthName((utf8String)"/C=us/O=MYCOMPANY/CN=mb");
  if (status) return status;

  status = JNH_set_IniLdapAuthPwd((utf8String)"/C=us/O=MYCOMPANY/CN=mb");
  if (status) return status;
  }

uint32 objid;
status = JNH_create_BootStrap(&objid);
if (status) return status;

if (status = JNH_reserve_object(objid)) return status;
status = JNH_set_MyName(objid, (utf8String)"/C=us/O=IBM/OU=Test/CN=Issuer CA"); // CA case
if (status) return status;

status = JNH_set_certreq_privkey_EE(objid, (utf8String)"id-dsa", 512);
if (status) return status;

if (status = JNH_BootStrap(objid, (utf8String)"MYPIN"));
  return status;

// Now verify that the server has status of svrSt_Bootstrap
// and, if it does, dump the CA info to a file so that
// it can be used during RA enrollment with the CA.

octetString *serialNumber;
utf8String *fingerprint;
status = JNH_GetStatus(&serverstatus);
if (serverstatus == svrSt_Bootstrap) {
  if (IniAmICA()) // output CA info
     if (status = JNH_CA_write_info(utf8String"ca.info")) return status;

  if (status = JNH_get_fingerprint(serialNumber, fingerprint)) return status;
  // Output fingerprint to a file, such as cafingerprint.out or rafingerprint.out.
}
....
```

Enrolling the RA with the CA

This process establishes a trust relationship between the CA and its client RA. This is necessary to avoid having a masquerading RA interacting with the CA. Similarly, the RA verifies that it is enrolling with the legitimate CA server. RA enrollment consists of programming to both the RA and the CA. An enrollment request is initiated by the RA, which then gets approved or rejected by the CA. Figure 4 on page 214 illustrates the flow sequence of programming the RA enrollment with the CA.

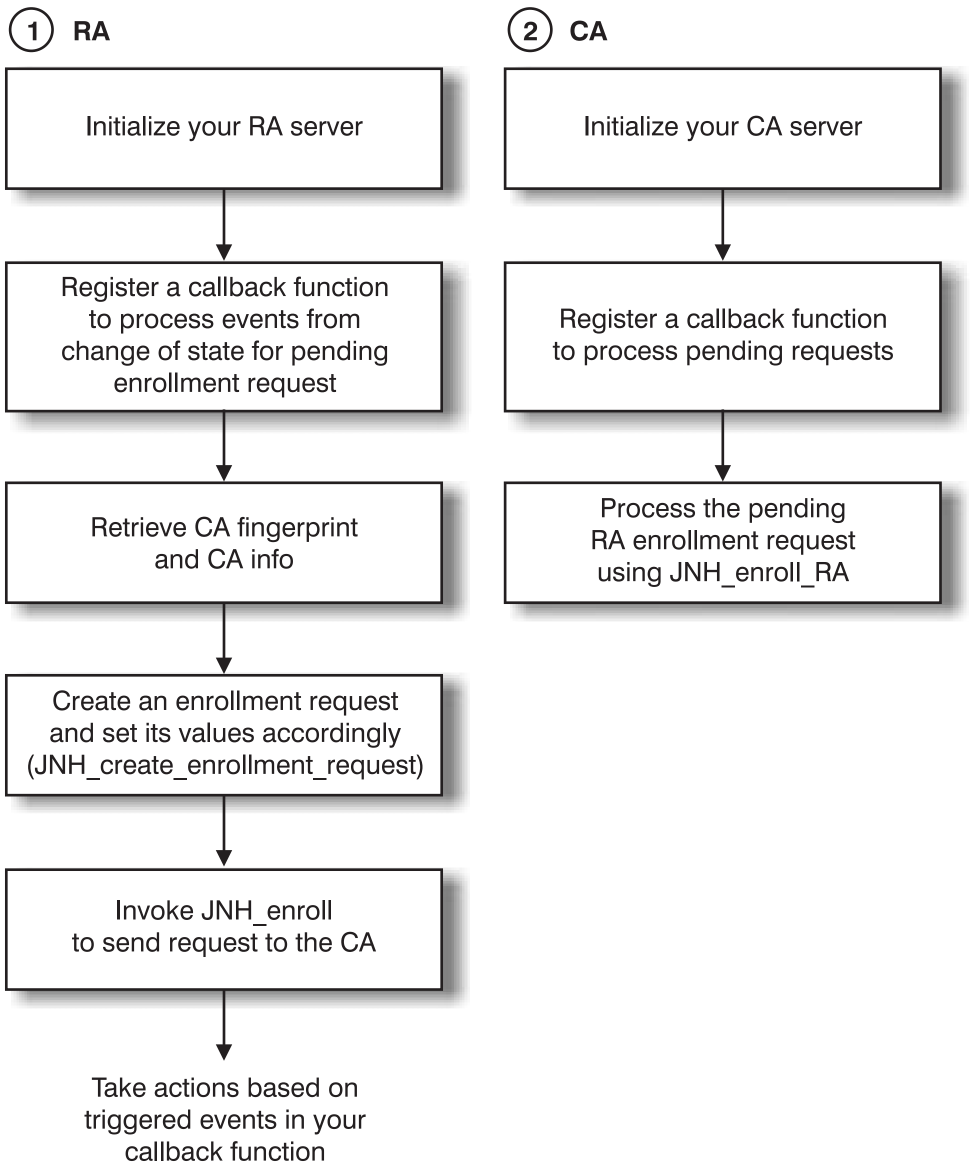
1
RA
Initialize your RA server
Register a callback function to process events from change of state for pending enrollment request
Retrieve CA fingerprint and CA info
Create an enrollment request and set its values accordingly (JNH_create_enrollment_request)
Invoke JNH_enroll to send request to the CA
Take actions based on triggered events in your callback function
2
CA
Initialize your CA server
Register a callback function to process pending requests
Process the pending RA enrollment request using JNH_enroll_RA

Figure 4. Overall steps for completing the RA enrollment with the CA

Sample RA enrollment code

This section contains code samples showing the RA issuing an enrollment request and the CA processing an enrollment request.

Initiating the RA enrollment request

```
int server_type = svrType_RA; // Initialize the RA run-time.
  uint32 status;
  if (status = JNH_start_server(server_type),(char*) "c:\pkix\myRA.ini") return status;

  if ((status = Init(svrType_RA)) != 0)
     return status;

  // Log in to RA server.
  status = JNH_server_login_pwd((utf8String)"raPin");

  // Register a callback function to handle events triggered
  // by state changes of an enrollment request object.
  if ((status = JNH_register_callbacks(&notify, &display))) return status;

   buffer_t buf;

  // Read in CA fingerprint file (cafinger.out) into buf.

  fp = fopen("c:\\cafingerprint", "rb");
  if (fp == NULL)
     return API_FILE_NOT_OPENED;

  c = getc(fp);
  while ((c != EOF) && (status == 0)){
     status = buff.append(c);
     if (status != 0) return status;
     c = getc(fp);
  }
  fclose(fp);

  // Get CA info in order to set enrollment request fields.
  status = JNH_get_CA_info((unsigned char *)filename, true, &caURL,
                      &subjName, &serialNumber);


  // Create the enrollment record.
  // It then computes a hash over the CA certificate and checks it against
  // the input CA fingerprint.

  uint32 reqId;
  status = JNH_create_enrollment_request((unsigned char *)"c:\\ca.info", true,
                                         buf.data, &reld)
  if (status) return status;

  if ((status = JNH_reserve_object(reqId))) return status;
  if ((status = JNH_set_certreq_issuer(objId, subjName))) return status;
  status = JNH_set_certreq_privkey_EE(objId, (utf8String)"rsaEncryption",512);
  if (status) return status;

  if (status = JNH_get_self_subjectKeyInfo(&numElements, &serialNumbers,
         &keyAlgorithms, &keyLengths))
     return status;
  if (numElements == 0)
     return API_INVALID_ARGUMENT;
  if (status = JNH_set_certreq_subjectKeyInfo(reqId, serialNumbers[0]))
     return status;


  status = JNH_save_object(objId);

  // Issue the enrollment.
  // Returns 10061 if it can't send the message to the CA.
  if ((status = JNH_enroll(reqId)))
     return status;
  ...;


void notify(uint32 id, const utf8String name, uint32 status)
{

  uint32 foo;

  foo = status & 0xffff0000;

  if (foo == ObjStRAEnrollWaitingForCAS)
     cout << "ra_enroll : RA enrollment request with ID = " << id << " is
     waiting for CA to approve"<< endl;
  else if (foo == ObjStRAEnrollActive)
```

```
     cout << "ra_enroll : RA enrollment request is active with ID = " << id
 << endl;

}
```

CA processing of the RA enrollment request

```
// Start the CA server, perform server login steps,
// and register a notify callback function as in above sample code.
// Also, retrieve the RA fingerprint by reading it from an input
// file, such as into a buffer rafgPrint visible to the notify function.

utf8String rafgPrint;

...;

// The notify function will process the enrollment request
// as in the sample below.


status = JNH_enroll_RA(objId, (const utf8String) x://localhost:829,
                              (utf8String) rafgPrint);


void notify(uint32 id, const utf8String name, uint32 status)
{
  uint32 foo = status & 0xffff0000;
  uint32 status;

  if (foo == ObjStCAEnrReqActive) {
    fprintf(stderr, "Enrollment Request is active, enrolling ra\n");
      utf8String fprint = NULL; // not checked until message protection task
    status = JNH_enroll_RA(id, (utf8String)"pkix://localhost:829", fprint);
    if (status) fprintf(stderr, "enroll RA failed with %d\n", status);
  }
  else { ....// perform other functions }

  ...;
  return;
}
```

Programming to the end entity

Certificate life cycle operations are generally initiated by the EE component of PKIX. A typical PKIX environment consists of one CA server, one RA server, and multiple EE images interacting with the RA. A preregistration process of subjects that are potential certificate holders needs to be performed early in the process. The resulting piece of information is a preregistration record that becomes the basis for an EE requesting a certificate. The preregistration step provides for a secure and a reliable means of verifying that the EE is acting in behalf of a legitimate subject and is requesting a certificate from a legitimate RA server.

An operation initiated at the EE might include an object store inquiry, a certificate request (for a new certificate), or a request to revoke an existing certificate.

The process of requesting a certificate by an EE is shown in Figure 5 on page 217 and is detailed in the sample code below. Although the preregistration process is programmed to the RA component, it is illustrated here as an integral part to such a process.

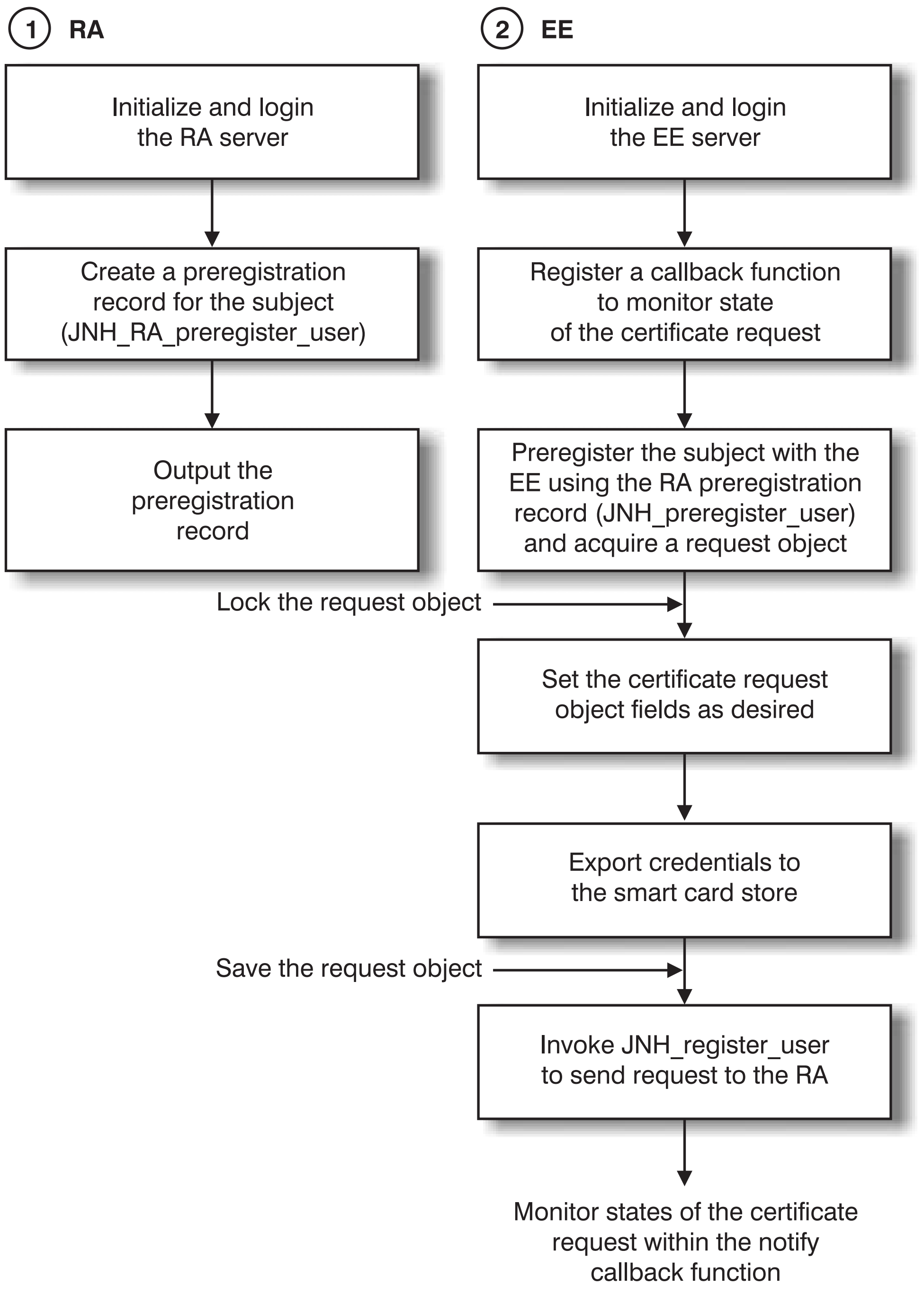
1 RA
2 EE
Initialize and login the RA server
Create a preregistration record for the subject (JNH_RA_preregister_user)
Output the preregistration record
Initialize and login the EE server
Register a callback function to monitor state of the certificate request
Preregister the subject with the EE using the RA preregistration record (JNH_preregister_user) and acquire a request object
Lock the request object
Set the certificate request object fields as desired
Export credentials to the smart card store
Save the request object
Invoke JNH_register_user to send request to the RA
Monitor states of the certificate request within the notify callback function

Figure 5. Programming steps for a certificate initial request

Sample EE certificate life cycle code

This section contains code samples showing preregistration at the RA and requesting a certificate.

Performing user preregistration at the RA

```
utf8String passwd = (unsigned char *) "myRapass";
  if (status = JNH_start_server(svrType_RA),(char *) "c:\pkix\myRA.ini") return status;
  if (status = JNH_server_login_pwd(passwd)) return status;
  utf8String CAName = NULL;
  if (status = JNH_INI_readString((unsigned char *const)"General",
                              (unsigned char *const)"Issuer1",
                              (unsigned char **) &CAName,
                              (unsigned char *const)"")) return status;

  utf8String name = (unsigned char *) "/C=us/O=myCo/OU=myName"; // Set to the subject name.
  utf8String pwd = (unsigned char *) "mypreregpass"; // Set to the preregistration password.

  utf8String buf;

  // Any value can be used instead of 0 in the following call.
  if (status = JNH_RA_preregister_user(CAName, name, 0, pwd, &buf)) return status;

  // Output buf into a file, such as preregrec.out.
```

Requesting a certificate

```
uint32 status;
  utf8String PreRegistrationRecord;
  // Read in from preregrec.out into PreRegistrationRecord.

  if (status = JNH_start_server(svrType_EE)) return status;
  if (status = JNH_register_callbacks(&notify, &display)) return status;
  uint32 *req_id;
  utf8String pwd = "mypreregpass";

  if (status = JNH_preregister_user(PreRegistrationRecord, pwd, &req_id)) return status);
  if (status = JNH_reserve_object(req_id)) return status;


  utf8String extn_id = ...;
  utf8String extn_value = "myName@myDomain.com";

  status = JNH_add_certreq_extension(SUBJECT_ALT_NAME, req_id, extn_id, extn_value, false);
  if (status) return status;

  if (status = JNH_set_certreq_privkey_EE(req_id, (utf8String)"id-dsa", 512))
  if (status) return status;

  if (status = JNH_save_object(id)) return status;

  utf8String tokenFile = "VSC:c:\\pkix\\data\\eetoken.fil"; // for example
  utf8String ecPwd;

  // In the following API, verify which VSC pin # is needed for the user/security officer.
  if (status = JNH_export_credential(req_id, ecPwd, tokenFile)) return status;
  if (status = JNH_register_user(req_id)) return status;
  // This sends a certificate request message to the RA.

  void notify(uint32 id, const utf8String name, uint32 status)
  {

  uint32 foo;
  bool found = false;

  foo = status & 0xffff0000;

  if (foo == ObjStEECertReqSubmittedS)
  {
    // Certificate request is submitted to the RA.
    ...;
  }

  if (foo == ObjStEECertIssued)
  {
     // A certificate was issued; take action accordingly.
     ...;
  }
  ...;
 }
```

Programming to the registration authority

Direct interactions with the RA are intended to approve or disapprove of a certificate request/revocation request or to inquire and modify the RA's object store. In addition, the RA handles the programming of subject registration. The following example demonstrates a way of programming to the RA during a certificate life cycle.

```
if (status = JNH_start_server(svrType_RA), (utf8String) "MyInifile" return status;
if (status = JNH_server_login_pwd((utf8String)"myRApass")) return status;
if (status = JNH_register_callbacks(&notify, &display)) return status;
if (status = JNH_list_object()) return status;
...;

void notify(uint32 id, const utf8String name, uint32 status)
{

  uint32 foo;

  foo = status & 0xffff0000;

  if (foo == ObjStRACertReqActive)
    status = JNH_authorize_registration(id, 0); // Authorizes the certificate.

  if (foo == ObjStRACertCAApproved)
    printStatus(id, name, status, "RA_CERT_CA_APPROVED");

  if (foo == ObjStRACertCARejected)
    printStatus(id, name, status, "RA_CERT_CA_REJECTED");

  if (foo == ObjStRARevReqActive)
    printStatus(id, name, status, "RA_REV_ACTIVE");

  if (foo == ObjStRARevReqWaitingForCAS)
    printStatus(id, name, status, "RA_REV_WAITING_FOR_CA");

  if (foo == ObjStRARevReqCAApproved)
    printStatus(id, name, status, "RA_REV_CA_APPROVED");

  if (foo == ObjStRACertReqRejecting)
    printStatus(id, name, status, "RA_CERT_REQ_REJECTING");

}
```

Programming to the certificate authority

The paradigm here is similar to what occurs with the RA. Programming to the CA is centered around retrieving CA active objects from the CA's object store when the objects arrive, and then processing them according to the message type that they represent. The following example shows how a certificate request is processed by the CA into an issued certificate.

```
status = JNH_start_server(svrType_CA, (utf8String) "MyInifile"));
status = JNH_server_login_pwd((utf8String) password);
status = JNH_register_callbacks(&notify, &display);
if (status = JNH_list_object()) return status;

void notify(uint32 id, const utf8String name, uint32 status)
{
  uint32 foo;

  foo = status & 0xffff0000;

  // States for CA.
  if (foo == ObjStCACertReqActive)
    if (status = JNH_create_certificate(id, 0)) return status;
...;
}
```

Header and library files

When compiling the JNH APIs, you must include one or more of the following header files:

asn1.h

asn1msg.h

asnbase.h

asnnames.h

asnstrng.h

jonah.h

jtime.h

objstates.h

platform.h

utctime.h

x509.h

jonahalg.h

When linking, you must include one or more of the following library files:

jonah.lib

ObjStore.lib

Asn1.lib

Misc.lib

JnhComms.lib

Policy.lib

wsock32.lib

BinBin.lib

jdl.lib

Note: Do not link with remJonah.lib

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
IBM
SecureWay

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

This program contains Standard Template Library (STL) software from Hewlett-Packard Company. Copyright (c) 1994.

- Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

This program contains Standard Template Library (STL) software from Silicon Graphics Computer Systems, Inc. Copyright (c) 1996–1999.

- Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided “as is” without express or implied warranty.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines the terms and abbreviations in this book that may be new or unfamiliar and terms that may be of interest. It includes terms and definitions from:

- The IBM Dictionary of Computing, New York: McGraw-Hill, 1994.
- The American National Standard Dictionary for Information Systems, ANSI X3.172–1990, American National Standards Institute (ANSI), 1990.
- The Answers to Frequently Asked Questions, Version 3.0, California: RSA Data Security, Inc., 1998.

Numbers

4758 PCI Cryptographic Coprocessor. A programmable, tamper-resistant cryptographic PCI-bus card offering high performance DES and RSA cryptographic processing. The cryptographic processes occur within a secure enclosure on the card. The card meets the stringent requirements of the FIPS PUB 140-1 level 4 standard. Software can run within the secure enclosure. For example, credit card transaction processing can use the SET standard.

A

Abstract Syntax Notation One (ASN.1). An ITU notation that is used to define the syntax of information data. It defines a number of simple data types and specifies a notation for identifying these types and for specifying values of these types. These notations can be applied whenever it is necessary to define the abstract syntax of information without curbing how the information is encoded for transmission.

access control list (ACL). A mechanism for limiting the use of a specific resource to authorized users.

ACL. Access control list.

action history. Accumulated events in the life cycle of a credential.

American National Standard Code for Information Interchange (ASCII). The standard code that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set uses a coded character set that consists of 7-bit coded characters (8 bits including a bit for parity checking). The character set consists of control characters and graphic characters.

American National Standards Institute (ANSI). An organization that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. It consists of producers, consumers, and general interest groups.

ANSI. American National Standards Institute.

API. Application program interface.

applet. A computer program that is written in Java and runs inside a Java-compatible Web browser. Also known as a Java applet.

application program interface (API). In PKIX, a functional interface that allows an application program that is written in a high-level language to use specific PKIX functions.

ASCII. American National Standard Code for Information Interchange.

ASN.1. Abstract Syntax Notation One.

asymmetric cryptography. Cryptography that uses different, asymmetric keys for encryption and decryption. Each user receives a pair of keys: a public key accessible to all, and a private key known only to the user. A secure transaction can occur when the public key and the corresponding private key match, enabling the decryption of the transaction. This is also known as key pair cryptography. *Contrast with* symmetric cryptography.

asynchronous communication. A mode of communication that does not require the sender and recipient to be present simultaneously.

audit trail. Data, in the form of a logical path, that links a sequence of events. An audit trail enables tracing of transactions or the history of a given activity.

authentication. The process of reliably determining the identity of a communicating party.

authorization. Permission to access a resource.

B

base64 encoding. A common means of conveying binary data with MIME.

Basic Encoding Rules (BER). The rules specified in ISO 8825 for encoding data units described in abstract syntax notation 1 (ASN.1). The rules specify the encoding technique, not the abstract syntax.

BER. Basic Encoding Rules.

business process objects. A set of code used to accomplish a specific registration operation, such as checking the status of an enrollment request or verifying that a public key was sent.

business process template. A set of business process objects that are run in a specified order.

browser. *See* Web browser.

browser certificate. A digital certificate is also known as a client-side certificate. It is issued by a CA through an SSL-enabled Web server. Keys in an encrypted file enable the holder of the certificate to encrypt, decrypt, and sign data. Typically, the Web browser stores these keys. Some applications permit storage of the keys on smart cards or other media. *See also* digital certificate.

bytecode. Machine-independent code that is generated by the Java compiler and run by the Java interpreter.

C

CA. Certificate authority.

CA certificate. A certificate your Web browser accepts, at your request, from a CA it does not recognize. The browser can then use this certificate to authenticate communications with servers that hold certificates issued by that CA.

CAST-64. A block cipher algorithm that uses a 64-bit block size and a 6-bit key. It was designed by Carlisle Adams and Stafford Tavares.

CCA. IBM Common Cryptographic Architecture.

CDSA. Common Data Security Architecture.

Common Data Security Architecture (CDSA). An initiative to define a comprehensive approach to security service and security management for computer-based security applications. It was designed by Intel, to make computer platforms more secure for applications.

certificate authority (CA). The software responsible for following an organization’s security policies and assigning secure electronic identities in the form of certificates. The CA processes requests from RAs to issue, renew, and revoke certificates. The CA interacts with the RA to publish certificates and CRLs in the Directory. *See also* digital certificate.

certificate extension. An optional feature of the X.509v3 certificate format that provides for the inclusion of additional fields in the certificate. There are standard extensions and user-defined extensions. Standard extensions exist for various purposes, including key and policy information, subject and issuer attributes, and certification path constraints.

certificate policy. A named set of rules that indicates the applicability of a certificate to a particular class of applications that have common security requirements. For example, a certificate policy might indicate whether a particular certification type allows a user to conduct transactions for goods within a given price range.

certificate profile. A set of characteristics that define the type of certificate wanted (such as SSL certificates or IPSec certificates). The profile aids in managing certificate specification and registration. The issuer can change the names of the profiles and specify characteristics of the desired certificate, such as the validity period, key usage, DN constraints, and so forth.

certificate revocation list (CRL). A digitally signed, time-stamped list of certificates that the certificate authority has revoked. The certificates in this list should be considered unacceptable. *See also* digital certificate.

certification. The process during which a trusted third party issues an electronic credential that vouches for an individual, business, or organizational identity.

CGI. Common Gateway Interface.

chain validation. The validation of all CA signatures in the trust hierarchy through which a given certificate was issued. For example, if a CA was issued its signing certificate by another CA, both signatures are validated during validation of the certificate that the user presents.

class. In object-oriented design or programming, a group of objects that share a common definition and therefore share common properties, operations, and behavior.

cleartext. Data that is not encrypted. *Synonym for* plaintext.

client. (1) A functional unit that receives shared services from a server. (2) A computer or program that requests a service of another computer or program.

client/server. A model in distributed processing in which a program at one site sends a request to a program at another site and waits for a response. The requesting program is called a client; the answering one is called a server.

code signing. A technique for signing executable programs with digital signatures. Code signing is designed to improve the reliability of software that is distributed over the Internet.

Common Cryptographic Architecture (CCA). IBM software that enables a consistent approach to cryptography on major IBM computing platforms. It supports application software that is written in a variety of programming languages. Application software can call on CCA services to perform a broad range of cryptographic functions, including DES and RSA encryption.

Common Gateway Interface (CGI). Standard method of transmitting information between Web pages and Web servers.

confidentiality. The property of not being divulged to unauthorized parties.

credential. Confidential information used to prove one's identity in an authentication exchange. In environments for network computing, the most common type of credential is a certificate that a CA has created and signed.

CRL. Certificate revocation list.

CRL publication interval. Set in the CA configuration file, the interval of time between periodic publications of the CRL to the Directory.

cross-certification. A trust model whereby one CA issues to another CA a certificate that contains the public key associated with its private signature key. A cross-certified certificate allows client systems or end entities in one administrative domain to communicate securely with client systems or end entities in another domain.

cryptographic. Pertaining to the transformation of data to conceal its meaning.

cryptography. In computer security, the principles, means, and methods for encrypting plaintext and decrypting encrypted text.

D

daemon. A program that carries out tasks in the background. It is implicitly called when a condition occurs that requires its help. A user need not be aware of a daemon, because the system usually spawns it automatically. A daemon might live forever or the system might regenerate it at intervals.

The term (pronounced *demon*) comes from mythology. Later, it was rationalized as the acronym DAEMON: Disk And Execution MONitor.

Data Encryption Standard (DES). An encryption block cipher, defined and endorsed by the U.S. government in 1977 as an official standard. IBM developed it originally. DES has been extensively studied since its publication and is a well-known and widely used cryptographic system.

DES is a symmetric cryptographic system. When it is used for communication, both the sender and receiver must know the same secret key. This key is used to encrypt and decrypt the message. DES can also be used for single-user encryption, such as to store files on a hard disk in encrypted form. DES has a 64-bit block size and uses a 56-bit key during encryption. It is was originally designed for implementation in hardware. NIST has recertified DES as an official U.S. government encryption standard every five years.

Data Storage Library (DL). A module that provides access to persistent data stores of certificates, CRLs, keys, policies, and other security-related objects.

decrypt. To undo the encryption process.

DEK. Document encrypting key.

DER. Distinguished Encoding Rules.

DES. Data Encryption Standard.

Diffie-Hellman. A method of establishing a shared key over an insecure medium, named after the inventors (Diffie and Hellman).

digital certificate. An electronic credential that is issued by a trusted third party to a person or entity. Each certificate is signed with the private key of the CA. It vouches for an individual, business, or organizational identity.

Depending on the role of the CA, the certificate can attest to the authority of the bearer to conduct e-business over the Internet. In a sense, a digital certificate performs a similar role to a driver's license or a medical diploma. It certifies that the bearer of the corresponding private key has authority to conduct certain e-business activities.

A certificate contains information about the entity it certifies, whether person, machine, or computer program. It includes the certified public key of that entity.

digital certification. *See* certification.

digital signature. A coded message added to a document or data that guarantees the identity of the sender.

A digital signature can provide a greater level of security than a physical signature. The reason for this is that a digital signature is not an encrypted name or series of simple identification codes. Instead, it is an encrypted summary of the message that is being signed. Thus, affixing a digital signature to a message provides solid identification of the sender. (Only the sender's key can create the signature.) It also fixes the content of the message that is being signed (the encrypted message summary must match the message content or the signature is not valid). Thus, a digital signature cannot be copied from one message and applied to another because the summary, or hash, would not match. Any alterations to the signed message would also invalidate the signature.

Digital Signature Algorithm (DSA). A public key algorithm that is used as part of the Digital Signature Standard. It cannot be used for encryption, only for digital signatures.

Directory. A hierarchical structure intended as a global repository for information related to communications (such as e-mail or cryptographic exchanges). The Directory stores specific items that are essential to the PKI structure, including public keys, certificates, and certificate revocation lists.

Data in the Directory is organized hierarchically in the form of a tree, with the root at the top of the tree. Often, higher level organizations represent individual countries, governments, or companies. Users and devices are typically represented as leaves of each tree. These users, organizations, localities, countries, and devices each have their own entry. Each entry consists of typed attributes. These provide information about the object that the entry represents.

Each entry in the Directory is bound with an associated distinguished name (DN). This is unique when the entry includes an attribute that is known to be unique to the real world object. Consider the following example DN. In it, the country (C) is US, the organization (O) is IBM, the organizational unit (OU) is Trust, and the common name (CN) is CA1.

```
C=US/O=vnet/OU=Trust/CN=CA1
```

Distinguished Encoding Rules (DER). Provides constraints on the BER. DER selects just one type of encoding from those that the encoding rules allow, eliminating all of the sender's options.

distinguished name (DN). The unique name of a data entry that is stored in the Directory. The DN uniquely identifies the position of an entry in the hierarchical structure of the Directory.

DL. Data Storage Library.

DN. Distinguished name.

document encrypting key (DEK). Typically, a symmetric encryption/decryption key, such as DES.

domain. *See* security domain *and* registration domain.

DSA. Digital Signature Algorithm.

E

e-business. Business transactions over networks and through computers. It includes buying and selling goods and services. It also includes transferring funds through digital communications.

e-commerce. Business-to-business transactions. It includes buying and selling goods and services (with customers, suppliers, vendors, and others) on the Internet. It is a primary element of e-business.

end entity. The subject of a certificate that is not a CA.

encrypt. To scramble information so that only someone who has the appropriate decryption code can obtain the original information through decryption.

encryption/decryption. Using the public key of the intended recipient to encipher data for that person, who then uses the private key of the pair to decipher the data.

enrollment attribute. An enrollment variable that is contained in an enrollment form. Its value reflects the information that is captured during the enrollment. The value of the enrollment attribute remains the same throughout the lifetime of the credential.

enrollment variable. *See* enrollment attribute.

extranet. A derivative of the Internet that uses similar technology. Companies are beginning to apply Web publishing, electronic commerce, message transmission, and groupware to multiple communities of customers, partners, and internal staff.

F

File Transfer Protocol (FTP). An Internet client/server protocol for use in transferring files between computers.

firewall. A gateway between networks that restricts the flow of information between networks. Typically, the purpose of a firewall is to protect internal networks from unauthorized use from the outside.

FTP. File Transfer Protocol.

G

gateway. A functional unit that allows incompatible networks or applications to communicate with each other.

H

HTML. Hypertext Markup Language.

HTTP. Hypertext Transaction Protocol.

HTTP server. A server that handles Web-based communications with browsers and other programs in a network.

hypertext. Text that contains words, phrases, or graphics that the reader can click with the mouse to retrieve and display another document. These words, phrases, or graphics are known as hyperlinks. Retrieving them is known as linking to them.

Hypertext Markup Language (HTML). A markup language for coding Web pages. It is based on SGML.

Hypertext Transaction Protocol (HTTP). An Internet client/server protocol for transferring hypertext files across the Web.

I

ICL. Issued certificate list.

IETF (Internet Engineering Task Force). A group that focuses on engineering and developing protocols for the Internet. It represents an international community of network designers, operators, vendors, and researchers. The IETF is concerned with the development of the Internet architecture and the smooth use of the Internet.

integrity. A system protects the integrity of data if it prevents unauthorized modification (as opposed to protecting the confidentiality of data, which prevents unauthorized disclosure).

integrity checking. The checking of audit records that result from transactions with external components.

internal structure. *See* schema.

International Standards Organization (ISO). An international organization tasked with developing and publishing standards for everything from wine glasses to computer network protocols.

International Telecommunication Union (ITU). An international organization within which governments and the private sector coordinate global telecommunication networks and services. It is the leading publisher of telecommunication technology, regulatory, and standards information.

Internet. A worldwide collection of networks that provide electronic connection between computers. This enables them to communicate with each other via software devices such as electronic mail or Web browsers. For example, some universities are on a network that in turn links with other similar networks to form the Internet.

intranet. A network within an enterprise that usually resides behind firewalls. It is a derivative of the Internet and uses similar technology. Technically, intranet is a mere extension of the Internet. HTML and HTTP are some of the commonalties.

IPSec. An Internet Protocol Security standard, developed by the IETF. IPSec is a network layer protocol, designed to provide cryptographic security services that flexibly support combinations of authentication, integrity, access control, and confidentiality. Because of its strong authentication features, it has been adopted by many VPN product vendors as the protocol for establishing secure point-to-point connections over the Internet.

ISO. International Standards Organization.

issued certificate list (ICL). A complete list of the certificates that have been issued and their current status. Certificates are indexed by serial number and state. This list is maintained by the CA and stored in the CA database.

ITU. International Telecommunication Union.

J

Java. A set of network-aware, non-platform-specific computer technologies developed by Sun Microsystems, Incorporated. The Java environment consists of the Java OS, the virtual machines for various platforms, the object-oriented Java programming language, and several class libraries.

Java applet. *See* applet. *Contrast with* Java application.

Java application. A stand-alone program that is written in the Java language. It runs outside the context of a Web browser.

Java class. A unit of Java program code.

Java language. A programming language, developed by Sun Microsystems, designed specifically for use in applet and agent applications.

Java Virtual Machine (JVM). The part of the Java run-time environment responsible for interpreting bytecodes.

K

key. A quantity used in cryptography to encipher or decipher information.

key pair. Corresponding keys that are used in asymmetric cryptography. One key is used to encrypt and the other to decrypt.

L

LDAP. Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP). A protocol used to access the Directory.

M

MAC. Message authentication code.

MIME (Multipurpose Internet Mail Extensions). A freely available set of specifications that allows the interchange of text in languages with different character sets. it also allows multimedia e-mail among many different computer systems that use Internet mail standards. For example, the e-mail messages may contain character sets other than US-ASCII, enriched text, images, and sounds.

modulus. In the RSA public key cryptographic system, the product (`n`) of two large primes: `p` and `q`. The best size for an RSA modulus depends on one's security needs. The larger the modulus, the greater the security. The current RSA Laboratories–recommended key sizes depend on the planned use for the key: 768 bits for personal use, 1024 bits for corporate use, and 2048 bits for extremely valuable keys like the key pair of a CA. A 768-bit key is expected to be secure until at least the year 2004.

N

National Language Support (NLS). Support within a product for differences in locales, including language, currency, date and time format, and numeric presentation.

National Security Agency (NSA). The official security body of the U.S. government.

NIST. National Institute of Standards and Technology, formerly known as NBS (National Bureau of Standards). It promotes open standards and interoperability in computer-based industries.

NLS. National language support.

nonce. A string that is sent down from a server or application, requesting user authorization. The user that is asked for authentication signs the nonce with a private key. The user's public key and the signed nonce are sent back to the server or application that requested authentication. The server then attempts to decipher the signed nonce with the user's public key. If the deciphered nonce is the same as the original nonce that was sent, the user is authenticated.

non-repudiation. The use of a digital private key to prevent the signer of a document from falsely denying having signed it.

NSA. National Security Agency.

O

object. In object-oriented design or programming, an abstraction encapsulating data and the operations associated with that data. *See also* class.

object identifier (OID). An administratively assigned data value of the type defined in abstract syntax notation 1 (ASN.1).

object type. The kind of object that can be stored in the Directory. For example, an organization, meeting room, device, person, program, or process.

ODBC. Open Database Connectivity.

Open Database Connectivity (ODBC). A standard for accessing different database systems.

Open Systems Interconnect (OSI). The name of the computer networking standards that the ISO approved.

OSI. Open Systems Interconnect.

P

PC card. Similar to a smart card, and sometimes called a PCMCIA card. This card is somewhat larger than a smart card and usually has a greater capacity.

PEM. Privacy-enhanced mail.

PKCS. Public Key Cryptography Standards.

PKCS #1. *See* Public Key Cryptography Standards.

PKCS #7. *See* Public Key Cryptography Standards.

PKCS #10. *See* Public Key Cryptography Standards.

PKCS #11. *See* Public Key Cryptography Standards.

PKCS #12. *See* Public Key Cryptography Standards.

PKI. Public key infrastructure.

PKIX. An X.509v3-based PKI.

PKIX certificate management protocol (CMP). A protocol that enables connections with PKIX-compliant applications. PKIX CMP uses TCP/IP as its primary transport mechanism, but an abstraction layer over sockets exists. This enables support for additional polling transports.

PKIX CMP. PKIX certificate management protocol.

plaintext. Unencrypted data. *Synonym for* cleartext.

policy exit. In a registration application, an organization-defined program that is called by the application. The rules specified in a policy exit apply the organization's business and security preferences to the enrollment process.

privacy. Protection from the unauthorized disclosure of data.

privacy-enhanced mail (PEM). The Internet privacy-enhanced mail standard, that the Internet Architect Board (IAB) adopted to provide secure electronic mail over the Internet. The PEM protocols provide for encryption, authentication, message integrity, and key management.

private key. The key in a public/private key pair that is available only to its owner. It enables the owner to receive a private transaction or make a digital signature.

Data signed with a private key can be verified only with the corresponding public key. *Contrast with* public key. *See also* public/private key pair.

protocol. An agreed-on convention for inter-computer communication.

proxy server. An intermediary between the computer that is requesting access (computer A) and the computer that is being accessed (computer B). Thus, if an end user makes a request for a resource from computer A, this request is directed to a proxy server. The proxy server makes the request, gets the response from computer B, and then forwards the response to the end user. Proxy servers are useful for accessing World Wide Web resources from inside a firewall.

public key. The key in a public/private key pair that is made available to others. It enables them to direct a transaction to the owner of the key or verify a digital signature. Data encrypted with the public key can be decrypted only with the corresponding private key. *Contrast with* private key. *See also* public/private key pair.

Public Key Cryptography Standards (PKCS). Informal inter-vendor standards developed in 1991 by RSA Laboratories with representatives from various computer vendors. These standards cover RSA encryption, the Diffie-Hellman agreement, password-based encryption, extended-certificate syntax, cryptographic message syntax, private-key information syntax, and certification syntax.

- PKCS #1 describes a method for encrypting data by using the RSA public key cryptosystem. Its intended use is in the construction of digital signatures and digital envelopes.
- PKCS #7 specifies a general format for cryptographic messages.
- PKCS #10 specifies a standard syntax for certification requests.
- PKCS #11 defines a technology-independent programming interface for cryptographic devices such as smart cards.
- PKCS #12 specifies a portable format for storing or transporting a user's private keys, certificates, miscellaneous secrets, and so forth.

public key infrastructure (PKI). A standard for security software that is based on public key cryptography. The PKI is a system of digital certificates, certificate authorities, registration authorities, certificate management services, and distributed directory services. It is used to verify the identity and authority of each party involved in any transaction over the Internet. These transactions might involve operations where identity verification is required. For example, they might confirm the origin of proposal bids, authors of e-mail messages, or financial transactions.

The PKI achieves this by making the public encryption keys and certificates of users available for authentication by a valid individual or organization. It provides online directories that contain the public encryption keys and certificates that are used in verifying digital certificates, credentials, and digital signatures.

The PKI provides a means for swift and efficient responses to verification queries and requests for public encryption keys. It also identifies potential security threats to the system and maintains resources to deal with security breaches. Lastly, the PKI provides a digital timestamping service for important business transactions.

public/private key pair. A public/private key pair is part of the concept of key pair cryptography (introduced in 1976 by Diffie and Hellman to solve the key management problem). In their concept, each person obtains a pair of keys, one called the public key and the other called the private key. Each person's public key is made public while the private key is kept secret. The sender and receiver do not need to share secret information: all communications involve only public keys, and no private key is ever transmitted or shared. It is no longer necessary to trust some communications channel to be secure against eavesdropping or betrayal. The only requirement is that public keys must be associated with their users in a trusted (authenticated) manner (for instance, in a trusted directory). Anyone can send a confidential message by using public information. However, the message can be decrypted only with a private key, which is in the sole possession of the intended recipient. Furthermore, key pair cryptography can be used not only for privacy (encryption), but also for authentication (digital signatures).

R

RA. Registration authority.

RA administrator. A user who has been authorized to access the RA Desktop, to administer certificates and requests for certificates.

RC2. A variable key-size block cipher, designed by Ron Rivest for RSA Data Security. *RC* stands for *Ron's Code* or *Rivest's Cipher*. It is faster than DES and is designed as a drop-in replacement for DES. It can be made more secure or less secure against exhaustive key search than DES by using appropriate key sizes. It has a block size of 64 bits and is about two to three times faster than DES in software. RC2 can be used in the same modes as DES.

An agreement between the Software Publishers Association (SPA) and the United States government gives RC2 special status. This makes the export approval process simpler and quicker than the usual cryptographic export process. However, to qualify for quick export approval a product must limit the RC2 key size to 40 bits with some exceptions. An additional string can be used to thwart attackers who try to precompute a large look-up table of possible encryptions.

registration authority (RA). The software that administers digital certificates to ensure that an organization’s business policies are applied from the initial receipt of an enrollment request through certificate revocation.

registration database. Contains information about certificate requests and issued certificates. The database stores enrollment data and all changes to the certificate data throughout its life cycle. The database can be updated by RA processes and policy exits, or by RA administrators.

registration domain. A set of resources, policies, and configuration options related to specific certificate registration processes. The domain name is a subset of the URL that is used to run the registration application.

repudiate. To reject as untrue; for example, to deny that you sent a specific message or submitted a specific request.

request ID. A 24- to 32-character ASCII value that uniquely identifies a certificate request to the RA. This value can be used on the certificate request transaction to retrieve the status of the request or the certificate that is associated with it.

RSA. A public key cryptographic algorithm that is named for its inventors (Rivest, Shamir, and Adelman). It is used for encryption and digital signatures.

S

schema. As relates to the Directory, the internal structure that defines the relationships between different object types.

Secure Electronic Transaction (SET). An industry standard that facilitates secure credit card or debit card payment over untrusted networks. The standard incorporates authentication of cardholders, merchants, and card-issuing banks because it calls for the issuance of certificates.

Secure Sockets Layer (SSL). An IETF standard communications protocol with built-in security services that are as transparent as possible to the end user. It provides a digitally secure communications channel.

An SSL-capable server usually accepts SSL connection requests on a different port than requests for standard HTTP requests. SSL creates a session during which the exchange signals to set up communications between two modems need to occur only once. After that, communication is encrypted. Message integrity checking continues until the SSL session expires.

security domain. A group (a company, work group or team, educational or governmental) whose certificates have been certified by the same CA. Users with certificates that are signed by a CA can trust the identity of another user that has a certificate signed by the same CA.

server. (1) In a network, a data station that provides functions to other stations; for example, a file server. (2) In TCP/IP, a system in a network that handles the requests of a system at another site, called a client/server.

server certificate. A digital certificate, issued by a CA to enable a Web server to conduct SSL-based transactions. When a browser connects to the server by using the SSL protocol, the server sends the browser its public key. This enables authentication of the identity of the server. It also enables encrypted information to be sent to the server. *See also* CA certificate, digital certificate, *and* browser certificate.

servlet. A server-side program that gives Java-enabled servers additional functionality.

SET. Secure Electronic Transaction.

SGML. Standard Generalized Markup Language.

SHA-1 (Secure Hash Algorithm). An algorithm that was designed by NIST and NSA for use with the Digital Signature Standard. The standard is the Secure Hash Standard; SHA is the algorithm that the standard uses. SHA produces a 160-bit hash.

sign. To use your private key to generate a signature. The signature is a means of proving that you are responsible for and approve of the message you are signing.

signing/verifying. To sign is to use a private digital key to generate a signature. To verify is to use the corresponding public key to verify the signature.

Simple Mail Transfer Protocol (SMTP). A protocol that transfers electronic mail over the Internet.

site certificate. Similar to a CA certificate, but valid only for a specific Web site. *See also* CA certificate.

smart card. A piece of hardware, typically the size of a credit card, for storing a user’s digital keys. A smart card can be password-protected.

S/MIME. A standard that supports the signing and encryption of e-mail transmitted across the Internet. *See* MIME.

SMTP. Simple Mail Transfer Protocol.

SSL. Secure Sockets Layer.

Standard Generalized Markup Language (SGML). A standard for describing markup languages. HTML is based on SGML.

symmetric cryptography. Cryptography that uses the same key for both encryption and decryption. Its security rests in the key — revealing the key means that anyone could encipher and decipher messages. The communication remains secret only as long as the key remains secret. *Contrast with* asymmetric cryptography.

symmetric key. A key that can be used for both encryption and decryption. *See also* symmetric cryptography.

T

target. A designated or selected data source.

TCP/IP. Transmission Control Protocol/Internet Protocol.

top CA. The CA at the top of a PKI CA hierarchy.

TP. Trust Policy.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of communication protocols that support peer-to-peer connectivity functions for local and wide area networks.

triple DES. A symmetric algorithm that encrypts the plaintext three times. Although many ways exist to do this, the most secure form of multiple encryption is triple-DES with three distinct keys.

trust domain. A set of entities whose certificates have been certified by the same CA.

trusted computer base (TCB). The software and hardware elements that collectively enforce an organization's computer security policy. Any element or part of an element that can effect security policy enforcement is security-relevant and part of the TCB. The TCB is an object that is bounded by the security perimeter. The mechanisms that carry out the security policy must be non-circumventable, and must prevent programs from gaining access to system privileges to which they are not authorized.

trust model. A structuring convention that governs how certificate authorities certify other certificate authorities.

tunnel. In VPN technology, an on-demand virtual point-to-point connection made through the Internet. While connected, remote users can use the tunnel to exchange secure, encrypted, and encapsulated information with servers on the corporate private network.

type. *See* object type.

U

Unicode. A 16-bit character set that is defined by ISO 10646. The Unicode character encoding standard is an international character code for information processing. The Unicode standard encompasses the principal scripts of the world and provides the foundation for the internationalization and localization of software. All source code in the Java programming environment is written in Unicode.

Uniform Resource Locator (URL). A scheme for addressing resources on the Internet. The URL specifies the protocol, host name or IP address. It also includes the port number, path, and resource details needed to access a resource from a particular machine.

URL. Uniform Resource Locator.

user authentication. The process of validating that the originator of a message is the identifiable and legitimate owner of the message. It also validates that you are communicating with the end user or system you expected to.

UTF-8. A transformation format. It enables information processing systems that handle only 8-bit character sets to convert 16-bit Unicode to an 8-bit equivalent and back again without loss of information.

V

VPN. Virtual Private Network.

Virtual Private Network (VPN). A private data network that uses the Internet rather than phone lines to establish remote connections. Because users access corporate network resources through an Internet Service Provider (ISP) rather than a telephone company, organizations can significantly reduce remote access costs. A VPN also enhances the security of data exchanges. In traditional firewall technology, message content can be encrypted, but the source and destination addresses are not. In VPN technology, users can establish a tunnel connection in which the entire information packet (content and header) is encrypted and encapsulated.

W

Web browser. Client software that runs on a desktop PC and enables the user to browse the World Wide Web or local HTML pages. It is a retrieval tool that provides universal access to the large collection of hypermedia material available in the Web and Internet. Some browsers can display text and graphics, and some can display only text. Most browsers can handle the major forms of Internet communication, such as FTP transactions.

Web server. A server program that responds to requests for information resources from browser programs. *See also* server.

World Wide Web (WWW). That part of the Internet where a network of connections is established between computers that contain hypermedia materials. These materials provide information and can provide links to other materials in the WWW and Internet. WWW resources are accessed through a Web browser program.

X

X.500. A standard for putting into effect a multipurpose, distributed and replicated directory service by interconnecting computer systems. Jointly defined by the International Telecommunications Union (ITU), formerly known as CCITT, and the International Organization for Standardization and International Electro-Chemical Commission (ISO/IEC).

X.509 certificate. A widely-accepted certificate standard designed to support secure management and distribution of digitally signed certificates across secure Internet networks. The X.509 certificate defines data structures that accommodate procedures for distributing public keys that are digitally signed by trusted third parties.

X.509 Version 3 certificate. The X.509v3 certificate has extended data structures for storing and retrieving certificate application information, certificate distribution information, certificate revocation information, policy information, and digital signatures.

X.509v3 processes create time-stamped CRLs for all certificates. Each time a certificate is used, X.509v3 capabilities allow the application to check the validity of the certificate. It also allows the application to determine whether the certificate is on the CRL. X.509v3 CRLs can be constructed for a specific validity period. They can also be based on other circumstances that might invalidate a certificate. For example, if an employee leaves an organization, their certificate would be put on the CRL.

Index

A

Abstract Syntax Notation One (ASN.1)
- abstract classes 207
- BER encoding 206
- class library hierarchy 207
- data types 205
- definition 205
- DER encoding 206
- header files 208
- PKIX data types 206
- X.509 certificate definition 205

application programming interfaces (APIs) 10
authorize registration request 34

B

background server 199
- initializing 91
- location, setting 198
- shutting down 199

basic constraints
- inquire 92
- set 168

BER encoding 206
boolean data type 206
bootstrap 36
- aborting 55
- MyName field, setting 186
- request 57

browser-based certificates
- certificate, retrieving 41, 145
- certificate request, removing 40, 144
- create revocation request object 141
- pre-registering a user 42, 148
- request revocation 142
- status 39, 143
- submitting a certificate request 43, 151

C

callbacks, register 152
- registering 10

certificate 3
- create 58
- creation 3
- customizing 4
- extensions 5
- life cycle 3
- publish 139
- revocation 3
- revoke 164
- setting fields 4
- storing 1
- validity period, setting 4

certificate authority (CA) 1
- certificate request, creating 15
- certificate requests, processing 21
- certificate store 1

certificate authority (CA) 1 *(continued)*
- CRL, creating 22
- definition 1
- description 21
- fingerprint, returning 70
- initializing 9
- list RAs 37
- responsibilities 1
- retrieving information about 68
- revocation requests, approving 21
- revocation requests, rejecting 22
- revoking certificates 21
- signing key, listing 38

certificate issuer
- inquire 95, 104
- set 172
- set in revocation request 189

certificate request 1
- basic constraints 92, 168
- CA certificate request, creating 15
- certificate subject 179
- definition 1
- end-user private key 98, 175
- expiration date 94, 170
- extensions 30, 122, 160
- issuer 95, 172
- key usage 96, 173
- selecting a key pair 114
- serial number 99
- starting date 100, 177
- status 101
- subject key algorithm 103
- subject name 102

certificate revocation list (CRL)
- create 59
- creating 22
- description 22
- publishing to the RA 140
- retrieving 161

certificate serial number
- inquire 99, 105
- inquire all 106
- set in revocation request 191

certificate starting date
- inquire 100
- set 177

certificate status
- inquire 101

certificate store 1
certificate subject name
- inquire 102
- set 179

control objects
- certificate requests 1
- certificate store 1
- revocation requests 1

create certificate 58

credentials
  exporting 66
  unlocking with password 166
cross certification
  CA signing key, listing 38
  CA signing key, setting 167
  cross-certification request, creating 136
  preregistration record, creating 146
  submitting a request 203

D

DER encoding 206
distinguished name, syntax 7
DN syntax 7

E

end entity
  applications 13
  certificate requests, creating 13
  certificate requests, submitting 14
  definition 1
  initializing 9
  key store 1
  revocation request, creating 15
end-user private key
  inquire 98
  set 175
error
  message 69
expiration date
  inquire 94
  set 170
export
  from a smart card 125
  private key and public certificate 130
extensions
  add 30
  adding to a certificate 8
  definition 5
  identifiers 5
  modifying 122
  removing 160
  requesting 5
  types 5
  values 5

F

fingerprint, returning CA or RA 70

H

hold instruction
  inquire 108
  set 193

I

importing to smart card 133
ini files
  initializing 83
  key and section, deleting 81
  keys, reading 84
  LDAP AuthName, retrieving 72
  LDAP AuthName, setting 181
  LDAP server information, retrieving 74
  LDAP server information, setting 183
  LDAP server password, retrieving 73
  LDAP server password, setting 182
  MyName field, retrieving value 71
  MyName field, setting 186
  RA URL, storing 202
  section, deleting 82
  sections, reading 86
  string, reading 87
  string, writing 90
  TCP host, retrieving 75
  TCP host, setting 184
  TCP port, retrieving 76
  TCP port, setting 185
  writing to a file 89
invalidity date
  inquire 109
  set 194

K

key pairs
  listing 117
  specifying in certificate request 114
key store 1
key store certificates
  expiration date 46
  issuer 48
  key usage 49
  retrieving from key store 115
  serial number 51
  starting date 52
  subject 54
key usage
  inquire 96
  set 173
key usage data type 206
keystore
  certificates 46
  description 2
  key identifiers 119
  subject names 119

N

naame syntax, distinguished 7

O

object
  deleting 63
  listing 118
  lock 163
  release 158

object *(continued)*
  reserve 163
  save 165
  state 79
  surrogates, listing 121
  unlock 158, 165
object identifier syntax 7
object store
  active objects 2
  indexing 2
  request identifiers 2
  states 2
  surrogate objects 2
octet string, releasing 159
octetString data type 206
OID syntax 7

P

preregister user 137
preregistration record
  create 149
private key
  inquire end-user private key 98
  set end-user private key 175
private keys, listing 117
Public Key Infrastructure
  ASN.1 data types 206
  components 1
  overview 1
  server applications, writing 9
public keys, listing 117
publish certificate 139

R

RA enrollment
  creating a request 60
  submitting a request 64
registration authority (RA) 1, 17
  certificate requests, processing 17
  definition 1
  fingerprint, returning 70
  initializing 9
  pre-registering users 17
  responsibilities 1
  revocation requests, processing 18
  URL, storing in .ini file 202
registration request 13
  authorize 34
  reject 155
  submit 154
reject registration 155
revocation 15
  authorize 35
  certificate 164
  reject 156
  request 162
revocation reason
  inquire 111
  set 196
revocation request
  canceling 45

revocation request *(continued)*
  certificate issuer, inquire 104
  certificate issuer, set 189
  certificate serial number, inquire 105
  certificate serial number, inquire all 106
  certificate serial number, set 191
  creating 15, 61
  creating for smart card certificate 62
  definition 1
  hold instruction, inquire 108
  hold instruction, set 193
  invalidity date, inquire 109
  invalidity date, set 194
  new 124
  requests, number of 113
  revocation reason, inquire 111
  revocation reason, set 196

S

self-signed certificate
  key information, returning 78
  key information, setting 180
  serial number, returning 77
  storing 44
server
  initializing 9
  request bootstrap 57
  starting 200
  status 80
service and support viii
smart card
  exporting from 125
  importing to 133
  listing key pairs 117
  revocation request 62
subject key algorithm
  inquire 103

U

uint32 data type 206
user registration 154
  preregister 137
  reject 155
  validate 204
utcDateTime data type 206
utf8String data type 206

V

validate registration 204

Y

year 2000 readiness viii

Program Number: 5697–F93

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.