Administrator's Guide

Netscape Certificate Management System

Version 4.2

151-09679-01
April 2000

Contents

Part 6 Policies

About This Guide

The *Administrator's Guide* explains how to administer Netscape Certificate Management System (CMS). It assumes that you have read the installation documentation and have successfully installed Certificate Management System.

This preface has the following sections:

What's in This Guide

This guide explains how to configure, customize, and maintain Certificate Management System, and use it for issuing and managing certificates to various end entities, such as web browsers (users), servers, Virtual Private Network (VPN) clients, and Cisco™ routers.

Who Should Read This Guide

This guide is intended for Certificate Management System administrators.

What You Should Already Know

This guide assumes that you

- Are familiar with the basic concepts of public-key cryptography and the Secure Sockets Layer (SSL) protocol.
  - SSL cipher suites
  - The purpose of and major steps in the SSL handshake
- Understand the concepts of intranet, extranet, and the Internet security and the role of digital certificates in a secure enterprise. These include the following topics:
  - Encryption and decryption
  - Public keys, private keys, and symmetric keys
  - Significance of key lengths
  - Digital signatures
  - Digital certificates, including various types of digital certificates
  - The role of digital certificates in a public-key infrastructure (PKI)
  - Certificate hierarchies

  If you are new to these concepts, we recommend you read the security-related documents available online at this URL:

  `http://docs.iplanet.com/docs/manuals/security.html`

  You may also refer to the security-related appendixes (D and E) of the accompanying manual, *Managing Servers with Netscape Console*.
- Are familiar with the role of Netscape Console in managing Netscape version 4.x servers. Otherwise, see the accompanying manual, *Managing Servers with Netscape Console*.
- Have read the *Netscape Certificate Management System Installation and Deployment Guide*.

Conventions Used in This Guide

The following conventions are used in this guide:

- `Monospaced font`

  This typeface is used for any text that appears on the computer screen or text that you should type. It's also used for filenames, functions, and examples.

  Example: `Server Root` is the directory where the CMS binaries are kept.

- *Italic*

  Italic type is used for emphasis, book titles, and glossary terms.

  Example: This control depends on the access permissions the *superadministrator* has set up for you.

- Text within "quotation marks"

  Cross-references to other topics within this guide.

  Example: For more information, see "Issuing a Certificate to a New User" on page 154.

- **Boldface**

  Boldface type is used for various UI components such as captions and field names, and the terminology explained in the glossary, which can be found in *Netscape Certificate Management System Installation and Deployment Guide*.

  Example:

  Rotation frequency. From the drop-down list, select the interval at which the server should rotate the active error log file. The available choices are Hourly, Daily, Weekly, Monthly, and Yearly. The default selection is Monthly.

- `Monospaced [ ]`

  Square brackets enclose commands that are optional.

  Example:

`PrettyPrintCert <input_file> [<output_file>]`

`<input_file>` specifies the path to the file that contains the base-64 encoded certificate.

`<output_file>` specifies the path to the file to write the certificate. This argument is optional; if you don't specify an output file, the certificate information is written to the standard output.

- `Monospaced <>`

  Angle brackets enclose variables or placeholders. When following examples, replace the angle brackets and their text with text that applies to your situation. For example, when path names appear in angle brackets, substitute the path names used on your computer.

  Example: Using Netscape Communicator 4.04 or later, enter the URL for the administration server:

  `http://<server_name>.<your_domain>.<domain>:<port_number>`

- /

  A slash is used to separate directories in a path. If you use the Windows NT operating system, you should replace / with \ in paths.

  Example: Except for the Security Module Database Tool, you can find all the other command-line utilities at this location:

  `<server_root>/bin/cert/tools`

- Sidebar text

  Sidebar text marks important information. Make sure you read the information before continuing with a task.

  Examples:

Note You can use Netscape Console only when Administration Server is up and running.

Caution A caution note documents a potential risk of losing data, damaging software or hardware, or otherwise disrupting system performance.

Unix Marks text that applies only to the Unix versions of Certificate Management System.

Windows NT Marks text that applies only to the Windows NT version of Certificate Management System.

Where to Go for Related Information

This section summarizes the documentation that ships with Netscape Certificate Management System, using these conventions:

- `<server_root>` is the directory where the CMS binaries are kept (which you specify during installation).
- `<instance_id>` is the ID for this instance of Netscape Certificate Management System (specified during installation).

Important Do not change the default location of any of the HTML files; they are used for online help. You may move the PDF files to another location.

The documentation set for Netscape Certificate Management System includes the following:

- *Managing Servers with Netscape Console* provides background information on basic cryptography concepts and the role of Netscape Console. For the HTML version, open this file: `<server_root>/manual/en/admin/help/contents.htm`
- *Netscape Certificate Management System Installation and Deployment Guide* describes how to plan for and install Netscape Certificate Management System. To access the installation and configuration information from within the CMS Installation Wizard, click any help button.

  The HTML version of this guide is located at `<server_root>/manual/en/cert/dep_gide/contents.htm`.

  The PDF version of this guide is located at `<server_root>/manual/en/cert/pdf/cms42install.pdf`.
- *Netscape Certificate Management System Administrator's Guide* (this guide) provides detailed reference information on CMS administration interfaces. To access this information from the CMS window within Netscape Console, click any help button.

  To view the HTML version of this guide, open this file: `<server_root>/manual/en/cert/adm_gide/contents.htm`

  To view the PDF version of this guide, open this file: `<server_root>/manual/en/cert/pdf/cms42adm.pdf`

- *Netscape Certificate Management System Agent's Guide* provides detailed reference information on CMS agent interfaces. To access this information from the Agent Services pages, click any help button.

  To view the HTML version of this guide, open this file: `<server_root>/cert-<instance_id>/web/agent/manual/agt_gide/contents.htm`.

  To view the PDF version of this guide, open this file: `<server_root>/manual/en/cert/pdf/cms42agent.pdf`

- End-entity help (online only, not printed) provides detailed reference information on CMS end-entity interfaces. To access this information from the end-entity pages, click any help button.

  To view the HTML version of this document, open this file: `<server_root>/cert-<instance_id>/web/ee/manual/ee_gide/contents.htm`

For a complete list of all documentation that ships with Netscape Certificate Management System, including documentation for Directory Server, see Documentation Summary, located at `<server_root>/manual/index.html`.

For the latest information about Netscape Certificate Management System, including current release notes, technical notes, and deployment information, check this site: `http://docs.iplanet.com/docs/manuals/cms.html`

1

Netscape Certificate Management System

Chapter

1

Introduction to Certificate Management System

This chapter introduces Netscape Certificate Management System (CMS), a highly configurable set of software components and tools for creating, deploying, and managing certificates. Based on open standards for certificate management, Certificate Management System leverages Netscape Directory Server and Netscape Console to provide a complete, scalable, high-performance certificate management solution for extranets and intranets.

Whether you are looking for a security solution for your enterprise or setting up an independent certificate authority (CA) service, Certificate Management System offers a robust, customizable, and scalable foundation for your public-key infrastructure (PKI).

The chapter has the following sections:

Overview

Certificate Management System provides a highly scalable, easily deployable certificate infrastructure for supporting encryption, authentication, tamper detection, and digital signatures in networked communications. It is based on open standards and protocols such as Public-Key Cryptography Standard (PKCS) #7, 10, 11, and 12, Secure Sockets Layer (SSL), Lightweight Directory Access Protocol (LDAP), and the X.509 certificate formats recommended by the International Telecommunications Union (ITU). Certificate Management System is highly customizable and configurable, permitting rapid integration with existing client and server software, customer databases, security systems, and authentication procedures.

You can use Certificate Management System to set up and manage your own public-key infrastructure or to deploy a public certification authority. Certificate Management System meets the needs of an enterprise, leveraging your existing enterprise resources and services, and will grow with your business needs to meet the demand of Internet-scale deployments.

With Certificate Management System, you can do the following operations:

- Process certificate requests from various end entities, such as web browsers, servers, routers, and virtual private network (VPN) clients, and issue certificates that conform to X.509 version 3 standard.
- Employ specific authentication methods for end-entity certificate enrollment, renewal, and revocation.
- Specify policy restrictions on certificate-related operations, such as certificate formulation, issuance, renewal, and revocation.
- Specify policy restrictions on key-related operations, such as archival and recovery of end users' encryption private keys.
- Revoke certificates, and maintain and publish a list of revoked certificates.
- Search for certificates issued by the server.
- Set up hierarchies of certificate authorities—multiple subordinate CAs chained up to a root CA. Note that Certificate Management System can chain under popular public CAs that are already pretrust in popular client and server products.

- Publish certificate information and the list of revoked certificates to an LDAP-compliant directory, such as Netscape Directory Server, and maintain this information.
- Publish the list of revoked certificates to an LDAP-compliant directory, a flat file, and an online validation authority.

Key Features

Certificate Management System has many core features:

Support for open standards

With its support for open standards, Certificate Management System gives organizations confidence that they will be able to communicate within a heterogeneous computing environment. Specifically, Certificate Management System does the following:

- Formulates, signs, and issues industry-standard X.509 version 3 public-key certificates; version 3 certificates include extensions that make it easy to include organization-defined attributes. This means that you can use these certificates for extranet and Internet authentication as well.

  For details on setting extensions in certificates, see "Overview of Extension-Specific Policy Modules" on page 554.
- Supports RSA public-key algorithm for signing and encryption, DSA public-key algorithm for signing, and MD2, MD5 and SHA-1 for hashing.
- Supports signature key lengths of up to 1024 bits (DSA) and 4096 (RSA) on both hardware and software tokens. For details, see Appendix E "Export Control Information" of *Netscape Certificate Management System Installation and Deployment Guide*.
- Supports multiple message formats, such as KEYGEN/SPAC, CRMF/CMMF, CRS/CEP/SCEP, and PKCS #10, for certificate requests. All requests are delivered to Certificate Management System over HTTP or HTTPS; in the case of CRS/CEP/SCEP protocol, the delivery method is always over HTTP.

- Supports certificate formats that encompass certificates for SSL-based client and server authentication, secure Multipurpose Internet Mail Extensions (S/MIME) message signing and encryption, object signing, VPN clients, and Cisco™ routers.
- Supports generation and publication of CRLs conforming to X.509 version 1 and 2. For more information, see "CRL Extension Modules" on page 763.
- Publishes certificates and certificate revocation lists (CRLs) to the any LDAP-compliant directory over LDAP and HTTP/HTTPS connections. For more information, see "Publishing of CRLs to an LDAP Directory" on page 724.
- Publishes certificates and CRLs to a flat file for importing into other resources. For example, the sample code for Flat File CRL and certificate publisher can be customized to store certificates and CRLs in an Oracle RDBMS™. For more information, see "Publishing of CRLs to Flat Files" on page 726.
- Publishes CRLs to an online validation authority (or OCSP responder), such as ValiCert Certificate VA™, enabling real-time verification of certificates by OCSP-compliant clients. For more information, see "Publishing of CRLs to an Online Validation Authority" on page 726.

Separate subsystems for certificate and key operations

Certificate Management System includes three servers, the *Certificate Manager*, *Registration Manager*, and *Data Recovery Manager*.

- The Certificate Manager functions as the certificate authority (CA); it is the entity named in the issuer field of a certificate. The Certificate Manager can sign and revoke certificates and generate CRLs. It can accept certificate requests directly from end entities and via Registration Managers to which it has delegated certain certificate management functions, such as authentication of an end entity. The Certificate Manager also maintains a database of issued certificates so that it can track renewal, expiration, and revocation.
- The Registration Manager is an optional component in the PKI; it is a subordinate server to which a Certificate Manager can delegate some certificate management functions. For example, a Registration Manager may perform tasks, such as end-entity authentication and formulation of the certificate request for the Certificate Manager.

- The Data Recovery Manager is an optional component in the PKI. It provides key archival and recovery services for end users' encryption private keys.

For an overview of these subsystems, see *Netscape Certificate Management System Installation and Deployment Guide*.

Single CA supports multiple registration authorities

Certificate Management System lets you separate the registration process from the certificate-signing process with the help of Registration Managers. You can run multiple Registration Managers remotely, all reporting to a single CA—a Certificate Manager—to verify user identities and process certificate signing requests. The remote Registration Managers forward their completed and approved requests to the Certificate Manager for it to sign and issue the certificate automatically.

The Certificate Manager's ability to support multiple Registration Managers makes it more scalable and also adds an extra layer of security for the CA. For example, you can set a policy that requires all clients to go through a remote Registration Manager, and then have the remote Registration Manager route all client requests to the Certificate Manager located inside a firewall.

For more information, see "Trusted Managers" on page 181.

Ability to function as both a root and a subordinate CA in a CA hierarchy

Certificate Management System can function as a root (or parent) CA (in which case the server signs its own CA signing key as well as other CA signing keys) enabling you to create your own CA hierarchy. You can also install the server to function as a subordinate CA (in which case the server gets its CA signing key signed by another CA) in an existing CA hierarchy.

Ability to function as a linked CA

Certificate Management System can function as a *linked CA*, chaining up to many third-party or public CAs for validation; this provides cross-company trust, so applications can verify certificate chains outside the company certificate hierarchy.

PKCS #11 hardware support for smart cards and crypto accelerators

Certificate Management System supports smart cards and crypto accelerators provided by various third-party vendors of PKCS #11 version 2.1-compliant products. For a complete list of vendors, check this site:

`http://www.iplanet.com/products/infrastructure/dir_security/cert_sys/index.html`

You can configure the server to use different PKCS #11 modules to generate and store key pairs (and certificates) for the Certificate Manager, Registration Manager, and Data Recovery Manager. Using hardware for key storage (especially for Certificate Manager and Data Recovery Manager key pairs) reduces the risk of key compromise, because hardware tokens don't reveal keys or provide means for them to be revealed, once the keys are generated in the hardware. Note that PKCS #11 hardware devices also provide key backup and recovery features for backup and recovery of the key material stored on the hardware token. Be sure to refer to PKCS #11 vendor documentation on this subject.

For information on configuring Certificate Management System to use hardware tokens for generating and storing its key pairs and certificates, see "Tokens for Storing Keys and Certificates" on page 235.

Support for Netscape client and server products; client independence for non-Netscape products

Certificates issued by Certificate Management System work with existing Netscape client and server products that support SSL. The certificates also work (out of the box) with a variety of non-Netscape, standards-compliant applications. For a complete list of these products, see the information available at this URL:

`http://www.iplanet.com/products/infrastructure/dir_security/cert_sys/index.html`

Highly scalable certificate data store

Certificate Management System uses a highly scalable, high-performance certificate storage facility—a preconfigured version of Netscape Directory Server 4.x that's automatically installed with Certificate Management System—enabling you to issue and manage a large number of certificates. For more information, see "Internal Database" on page 163.

Flexible end-entity registration services framework

The registration services framework for end entities includes the most commonly expected PKI features: manual, directory-based, and directory- and PIN-based enrollment; certificate-authenticated renewals and revocations (based on SSL client authentication); certificate life-cycle operations that include automated certificate renewal and expiration notifications. These features are available out of the box for both Certificate Manager and Registration Manager.

- For information on enrollment, renewal, and revocation operations, see "Introduction to Authentication" on page 305.
- For information on automated notifications, see "Introduction to Job Scheduling and Notifications" on page 433.

Built-in plug-in modules for authentication, policy, job scheduling, and LDAP publishing

Certificate Management System simplifies the details involved in certificate issuance and management with its built-in, configurable, and extensible authentication, policy, job scheduling, and publishing components. Each of these components come with a set of default modules that enable you to configure Certificate Management System for your PKI requirements. For example, you can configure policy modules to determine the outcome of operations, such as certificate formulation (extensions, signing algorithm, key length, validity period, and so on), issuance, renewal, and revocation.

- For information authentication modules, see Chapter 10, "Authentication Modules for End-Entity Enrollment."
- For information job modules, see Chapter 14, "Introduction to Job Scheduling and Notifications."
- For information policy modules, see Chapter 17, "Constraints-Specific Policy Modules" and Chapter 18, "Extension-Specific Policy Modules."

- For information publishing modules, see Chapter 21, "Modules for Publishing Certificates and CRLs."

Single administration point achieved via LDAP-compliant directory integration

Certificate Management System works seamlessly with any LDAP-compliant directory services for easy distribution of certificates and CRLs, thus lowering the cost of information management. The shared directory architecture enables you to manage users, including their security credentials and other shared data, at a single place. Certificate Management System can do the following:

- Authenticate users based on the information that exists in the LDAP directory.
- Integrate certificate-related information with the user and group information that exists in the LDAP directory.
- Automatically publish certificates (when they are issued) and CRLs (when created or on a periodic basis) to the LDAP directory, from which they can be easily distributed to clients and servers.
- Automatically delete expired and revoked certificates from the directory.
- Connect to the directory using password-based (basic) or certificate-based (in the context of LDAP over SSL) authentication using a digital certificate.

Supports certificate generation for dual key pairs—separate key pairs for signing and encrypting mail messages

To support separate key pairs for signing and encrypting data, Certificate Management System supports generation of dual certificates for end entities capable of generating dual key pairs. If a client makes a certificate request for dual key pairs, the server issues two separate certificates.

For more information, see "Clients That Can Generate Dual Key Pairs" on page 1116.

Bundles a client, Netscape Personal Security Manager, that can generate dual key pairs

Certificate Management System bundles Netscape Personal Security Manager, which when plugged into Netscape Communicator version 4.7 or later enables it to support the CMC protocol and generate dual key pairs. Personal Security Manager is a standards-based, client-independent application that performs PKI operations on behalf of Communicator 4.7 and other applications. Personal Security Manager also provides advanced cryptographic capabilities for applications that connect with Certificate Management System and greatly simplifies PKI operations for end users. In particular, Personal Security Manager allows system administrators to take advantage of the following Certificate Management System features:

- One-click issuance of certificates.
- Forced certificate backup by end users after certificate issuance.
- Issuance and management of dual encryption and signing email certificates.
- Automatic storage of encryption private keys with a key recovery authority, such as the Data Recovery Manager, at the time a certificate is issued.
- Automatic revocation checking each time Personal Security Manager verifies a certificate.

For more information about Personal Security Manager, check this site:

`http://www.iplanet.com/products/infrastructure/dir_security/cert_sys/index.html`

During CMS installation, Personal Security Manager (version 1.1) files are placed here:

`<server_root>/psm11`

You can also download the latest version of Personal Security Manager from this site:

`http://www.iplanet.com/downloads/download/index.html`

Note that Personal Security Manager works only with Netscape Communicator, version 4.7 or later, which can be downloaded from this site:

`http://home.netscape.com/download/`.

Key archival and recovery for encryption private keys

If your organization uses S/MIME to encrypt mail messages, you can use the key archival feature offered by Certificate Management System to back up users' encryption private keys. This feature is useful when a key becomes unavailable—as, for instance, in the following cases:

- An employee loses an encryption private key (for example after a disk crash or by forgetting the password to the key file) and is unable to read previously encrypted data.
- An employee leaves the company, and company officials need to perform an audit that requires gaining access to the employee's encrypted data.

For more information, see "Setting Up Key Archival and Recovery Process" on page 1133.

Encrypted key storage and password-protected recovery

Certificate Management System stores users' encryption private keys in an encrypted key repository. Keys can be retrieved only by authorized key recovery agents. The key repository is encrypted using a Data Recovery Manager's storage private key, which is protected with one or more recovery agents' passwords. Only these designated recovery agents can authorize and initiate a key recovery process.

For more information, see "Where the Keys are Stored" on page 1119.

Extensive audit and log records for detection of tampering

Certificate Management System maintains audit trails for all events—certificate requests and issuance, revocation requests, CRL publication, and so on. These audit records enable you to detect any unauthorized access or activity. In addition, extensive system and error logs record various events and system errors so that you can monitor and debug the system. All log records are stored in your local file system for quick and easy retrieval.

For more information, see "Logs" on page 1043.

Supports signing of log files for tamper detection

Certificate Management System allows you to sign log files digitally before archiving them or distributing them for audit purposes. This feature enables you to check whether the log files were tampered with after being signed.

For more information, see "Signing Log Files" on page 1075.

Java SDK extension mechanism for customization

The Java-based software development kit (SDK) provided with Certificate Management System includes APIs for customizing different aspects of the system. You can write the following custom modules:

- Authentication—for authenticating end entities during certificate enrollment.
- Policy—for setting the rules applied by the individual subsystems.
- Jobs—for PKI-related jobs that run with the individual systems.
- Mapper and publisher classes—for publishing certificates and CRLs to an LDAP-compliant directory, flat file, and an OCSP responder.
- Gateway for end-entity and agent forms or HTTP interfaces—for customizing end-entity and agent interfaces.

For information about writing custom modules, check the CMS software design kit (SDK) and samples package installed at this location:

```
<server_root>/cms_sdk
```

For information about customizing end-entity and agent interfaces, see "Customizing End-Entity and Agent Interfaces" on page 905.

Easy migration path from Netscape Certificate Server 1.0x

Certificate Management System provides an easy migration path from Netscape Certificate Server 1.0x. The server includes a command-line-based migration tool that extracts the contents of a Certificate Server 1.0x database, including keys and certificates, and puts them in three platform-independent files. You then import the contents of these files into the internal database of Certificate Management System.

You can use the tool to migrate data to Certificate Management System versions 4.1 or 4.2. For more information about the tool, refer to Appendix A "Migrating from Certificate Server 1.x" of *Netscape Certificate Management System Installation and Deployment Guide*.

Easy upgrade from previous versions of Certificate Management System

Certificate Management System version 4.2 provides an easy upgrade path from its previous version, 4.1.

Easy, GUI-based server installation and management

An installation wizard automates the installation and initial configuration process, helping you install Certificate Management System quickly and easily. Then after installation, you can locally or remotely administer Certificate Management System from various computers on your network (using the encryption, message integrity, and authentication services of SSL) with the help of an administration interface called the Certificate Management System window or the CMS window.

For more information, see "The CMS Window" on page 71.

System Architecture

Certificate Management System comprises three servers, or *main subsystems*, and a number of system-level components that are shared by these servers. The main subsystems are:

- Certificate Manager
- Registration Manager
- Data Recovery Manager

The system-level components (also referred to as *subsystems*) are:

- Authentication
- Policy
- Job scheduling and notification

- Logging
- LDAP publishing
- Internal database

Figure 1.1 illustrates the high-level architecture of Certificate Management System.

Figure 1.1 Certificate Management System architecture

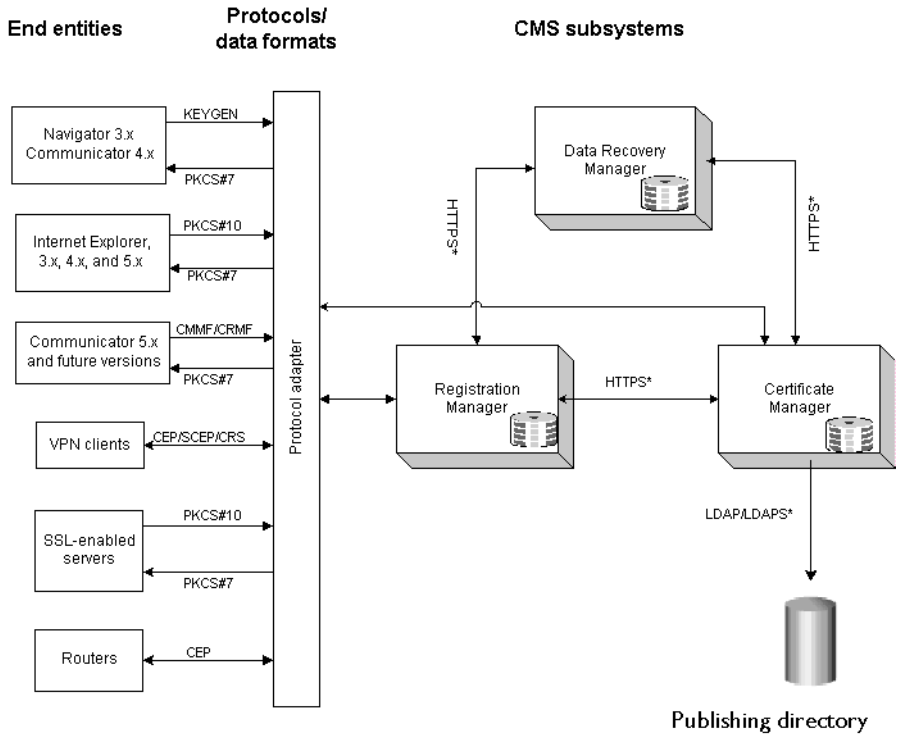


Table 1.1 gives an overview of each component's tasks. For detailed explanation for various components and how they interact, see the *Netscape Certificate Management System Installation and Deployment Guide*.

Table 1.1 Certificate Management System components

Component/Subsystem	Description
End entities	Various end entities that can request certificates from Certificate Management System: • Web browsers, such as Netscape Navigator version 3.x; Netscape Communicator versions 4.x and later; Microsoft Internet Explorer versions 3.x, 4.x, and 5.x • SSL-enabled servers, such as Netscape Administration, Directory, and Enterprise Servers • Routers, such as Cisco routers • Virtual private network clients, such as Aventail™, KyberPass™, and RedCreek™ For details on supported end-entity protocols, see "How Client Type Determines the End-Entity Interface" on page 897.
End-entity protocols	Various protocols that Certificate Management System supports for allowing end-entity interaction with the server. For details, see "Certificate Request Formats Specific to End Entities" on page 898.
Certificate Manager	Functions as the certificate authority (CA); it is the entity named in the issuer field of a certificate. Exposes a number of interfaces used by protocol adapters and other CMS subsystems or components to perform the fundamental tasks of certificate management; it is a service that signs and revokes certificates and generates CRLs. Accepts certificate requests directly from end entities as well as from Registration Managers to which it has delegated certain certificate management functions, such as authentication of an end entity.
Registration Manager	Functions as a full-fledged, remote registration front end to a Certificate Manager, enforcing policies (defined by the Policy engine) on certificate issuance, renewal, and revocation requests, key updates and recovery, and similar functions. Multiple Registration Managers can report to a single Certificate Manager (CA).
Data Recovery Manager	Handles encryption private key operations, such as key archival and recovery.
Internal database	An LDAP-compliant persistent storage system built into Certificate Management System. This database is a preconfigured version of Netscape Directory Server (version 4.x) installed transparently at the time of CMS installation.

Table 1.1 Certificate Management System components (Continued)

Component/Subsystem	Description
Publishing directory	Used for publishing certificates and CRLs. The Certificate Manager can publish to any LDAP-compliant directory (such as Netscape Directory Server 3.x and 4.x) used by organizations to maintain corporate data in a single place.

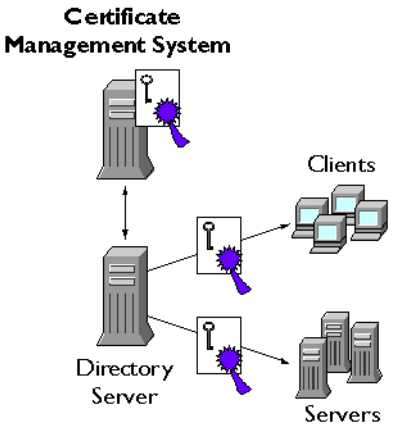
LDAP Directory Integration

Certificate Management System can function very closely with an LDAP-compliant directory, such as Netscape Directory Server, that organizations typically use to maintain corporatewide data about user and group accounts and other network resources. You can set up Certificate Management System to automatically publish certificate information and CRLs to a directory. The advantage of publishing certificates and CRLs to the directory is multifold:

- You can keep users' certificate-related information with the rest of the user information. This way, when you update the user information, the certificate-related information automatically gets updated. For example, when you delete a user entry, the security credentials of that user automatically gets deleted from the directory.
- If you are using S/MIME-enabled clients (for example, Netscape Communicator), publishing all certificates to a central directory enables your users to import others' certificates from the global directory.

For more information on setting up Certificate Management System to publish certificates and CRLs, see "Publishing" on page 713.

Figure 1.2 Seamless integration with any LDAP-compliant directory



Seamless integration with any LDAP-compliant directory (see Figure 1.2) makes possible the following:

- Corporate IS organizations can generate and manage certificates as an integral part of their user and group management.
- Independent CAs can issue and manage certificates to their users listed in any LDAP-compliant directory.

How the Main Subsystems Function

You can install all three subsystems, the Certificate Manager, Registration Manager, and Data Recovery Manager as standalone servers. You can also install the Data Recovery Manager with a Certificate Manager or Registration Manager—that is, the same instance of Certificate Management System will include both the servers. You cannot install a Registration Manager in the same instance as that of a Certificate Manager.

- If you installed the subsystems standalone, then you need to know whether they are in the same server group or in separate server groups; to understand server groups, see “Netscape Administration Server” on page 66.

— If the subsystems are in the same server group, then they use the same instance of Netscape Administration Server installed for that group. Administration Server enables you to launch the GUI-based administrative interface, Netscape Console, for both the instances. In this case, the URL for logging in to Netscape Console is the same for both the instances.

— If the subsystems are installed under separate server groups, then they use separate instances of Netscape Administration Server installed for those groups. In this case, the URL for logging in to Netscape Console will be different for the subsystems.

Note that when installed standalone, each subsystem has its own configuration file and an administrative interface, the CMS window, which can be launched within Netscape Console. The subsystem also has its own Directory Server instance (identified as the internal database) for storing and retrieving persistent objects such as certificates and keys.

- If you installed two subsystems in a single instance, then they both share the same configuration file and use the same internal database for storing and retrieving persistent objects such as certificates and keys. You can administer both the servers through a common interface, the CMS window, which can be launched within Netscape Console. In addition, the subsystems also share various supporting services, such as the template manager and logging.

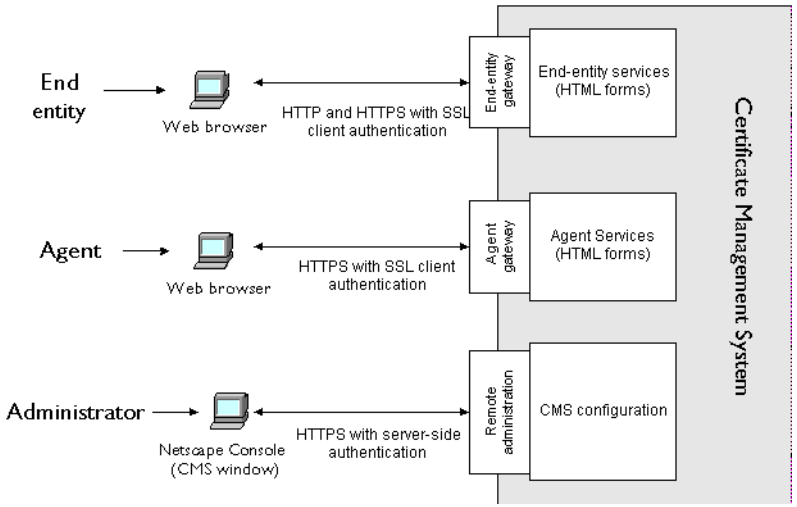
For information on Netscape Console and the CMS window, see "Administration Tasks and Tool" on page 63.

Installing the subsystems standalone requires you to do certain additional configurations so they can communicate with each other. For example, if you installed Registration Managers, you need to connect them to a Certificate Manager and optionally to a Data Recovery Manager. A Registration Manager connected to a Certificate Manager functions as a trusted front end to the Certificate Manager by receiving end-entity requests, authenticating them, and forwarding them to the Certificate Manager for signing. After receiving a response from the Certificate Manager, the Registration Manager notifies the end entity of the results. Similarly, a Certificate Manager or Registration Manager connected to a Data Recovery Manager handles end users' requests related to the archival of encryption private keys. For more information on connecting subsystems, see "Trusted Managers" on page 181.

Entry Points for Various Types of Users

Certificate Management System provides entry points for various kinds of user interaction.

Figure 1.3 Entry points for different types of users



As illustrated in Figure 1.3, the server provides three separate user entry points; each entry point addresses the needs of a specific user type. This is explained in Table 1.2.

Table 1.2 Certificate Server user-entry points

User	Component/ Tool	CMS interface	Description
End entity	Web browser	End Entity Services	This interface provides the general front end for end-entity interactions with the server. Through this interface, the Certificate Manager or Registration Manager serves the appropriate HTML forms for end-entity operations (the Data Recovery Manager does not have an end-entity interface). These include forms for certificate enrollment, retrieval, query, renewal, import, and revocation.

Table 1.2 Certificate Server user-entry points (Continued)

User	Component/ Tool	CMS interface	Description
Agent	Web browser	Agent Services	This interface provides the general front end for agent interactions with the server. Through this interface, a Certificate Manager, Registration Manager, or Data Recovery Manager serves the appropriate HTML forms for agent tasks. Accessing Agent Services is a privileged operation; agents must use designated certificates for SSL client authentication to Certificate Management System.
Administrator	Netscape Console (CMS window)	Remote administration	The remote administration interface supports a GUI-based administration tool called Netscape Console that provides the general administration and management interface for Certificate Management System. Administrators can use this tool to perform day-to-day operational and managerial duties, such as changing the server configuration, stopping and restarting the server, requesting and installing certificates, managing resources (certificates and requests), and setting up privileged-user information and associated access controls. The CMS window can only be launched from within Netscape Console. Accessing this window is a privileged operation requiring a password-based authentication to Certificate Management System.

Chapter

2

Administration Tasks and Tool

In administering Netscape Certificate Management System (CMS), you perform server-specific tasks such as starting, stopping, and restarting the server; changing configuration; configuring certificate issuance and management policies; adding or modifying privileged-user and group information; setting up authentication mechanisms for users who may request services from the server; performing routine server maintenance tasks; monitoring logs; and backing up server data.

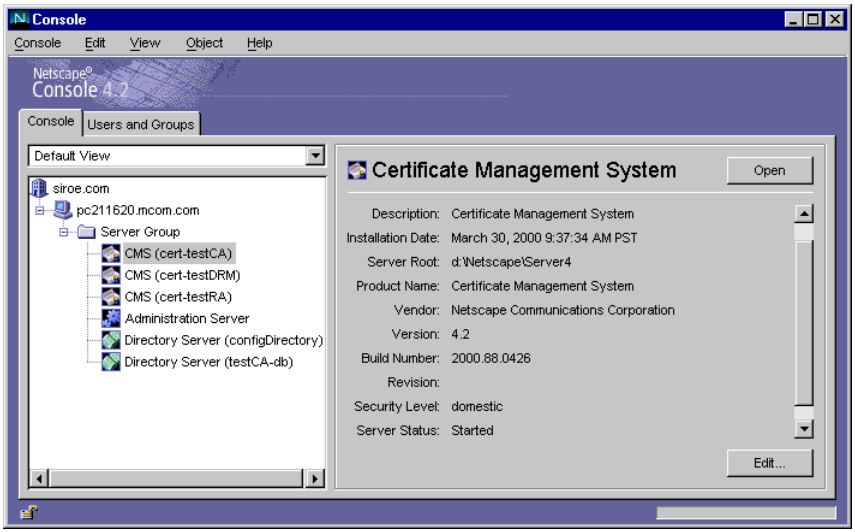
To enable system administrators to accomplish these server-specific tasks quickly and easily, Certificate Management System provides a GUI-based administration tool, called the CMS window, within Netscape Console. This chapter provides an overview of both Netscape Console and the CMS window.

Note You can use Netscape Console for managing various network resources. However, this chapter's focus is on using Netscape Console for CMS administration. For complete information about Netscape Console, see *Managing Servers with Netscape Console*, which is included with the CMS documentation.

Netscape Console

Netscape Console is a stand-alone Java application that provides a GUI-based front end to all network resources registered in an organization's configuration directory. This unified administration interface (shown in Figure 2.1) simplifies network administration by supplying access points to all Netscape version 4.x server instances installed across a network. Similarly, it simplifies basic user and group management by providing a unified administration interface to the user directory.

Figure 2.1 Main Netscape Console window, with a CMS instance selected in the Console tab



Console Tab

For any given instance of Netscape Console, the limits of the network it can administer are defined by the set of resources whose configuration information is stored in the same configuration directory—that is, the maximum set of hosts and servers that can be monitored from Netscape Console. The *superadministrator* (the person who manages the configuration directory) can set access permissions on all network resources registered in the configuration directory. Thus, for a given administrator using Netscape Console, the actual number of visible servers and hosts may be fewer, depending on the access permissions that the administrator has.

The Console tab displays all servers registered in a particular configuration directory, giving you a consolidated view of all the server software and resources under your control. What you control is determined by the access permissions the superadministrator has set up for you.

From this view you can perform tasks across arbitrary groups or a cluster of servers in a single operation. In other words, you can use the Console tab to manage a single server or multiple servers that are installed on different ports on one machine. Also, you can access individual server windows (or administration interfaces) by double-clicking the icons for the corresponding server instance entries (SIEs).

With the exception of Certificate Management System, all server instances displayed on the Console tab store their configuration information in the same configuration directory. For security purposes, Certificate Management System uses file-based configuration which is stored locally on the host system; during installation, the server registers only its SIE in the configuration directory. For details about this file, see "Configuration" on page 81.

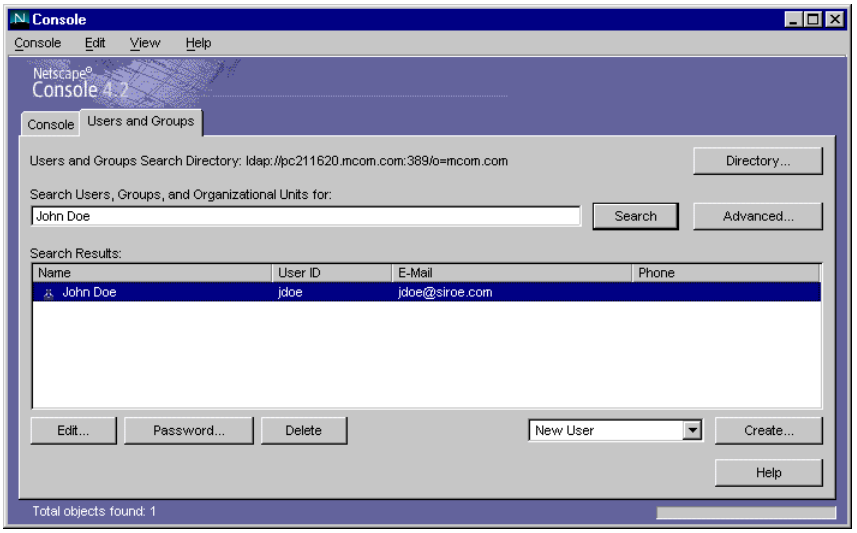
You can accomplish various CMS-specific tasks from the Console tab:

- Install multiple instances of Certificate Management System.
- Remove an instance of Certificate Management System from a system or host.
- Clone an instance of Certificate Management System.
- Set access permissions for Certificate Management System.
- Migrate configuration information from one version of Certificate Management System to another.
- Launch the Administration Server window (so that you can configure an Administration Server instance for administering Certificate Management System).
- Launch the CMS window.

Users and Groups Tab

The Users and Groups tab (shown in Figure 2.2) manages user accounts, group lists, and access control information for individual users and groups. All applications registered within the Netscape Console framework share core user and group information in the user directory, which typically is a global directory for corporatewide user data.

Figure 2.2 Users and Groups tab of Netscape Console



From this tab, you can accomplish various user- and group-specific tasks, such as these:

- Add, modify, and delete user and group information in the user directory.
- Search for specific user and group entries in the user directory.

Netscape Administration Server

Netscape Administration Server is a web-based (HTTP) server that enables you to configure all your Netscape version 4.x servers, including Certificate Management System, through Netscape Console. Administration Server (and the configuration directory) must be running before you can configure any of these

servers. It is included with all Netscape servers and is installed when you install your first server in a *server group*. A server group refers to servers that are installed in a server root directory and that are managed by a single instance of Netscape Administration Server.

You access Administration Server by entering its URL in the Netscape Console login screen. This URL is based on the computer host name and the port number you chose when you installed Certificate Management System. The format for the URL looks like this:

```
http://<machine_name>.<your_domain>.<domain>:<port_number>
```

Whenever you try to gain access to Administration Server, you will be prompted to authenticate yourself to the configuration directory by entering your user ID and password. These are the *administrator* user name and password that you specified when you installed Certificate Management System (or the first server in the server group) and Administration Server on your computer. Once Administration Server is running, you can use Netscape Console to administer all servers in that group, including Certificate Management System.

For complete details about Netscape Administration Server, see *Managing Servers with Netscape Console*. To locate an online version of this book, go to `<server_root>/manual/index.html`.

Starting Administration Server

The CMS installation program automatically starts the instance of Administration Server that you identified during installation for monitoring Certificate Management System. If you stopped Administration Server after installation, you must start it before you can administer Certificate Management System from the CMS window.

You can start the server from Netscape Console, the command line, or the Windows NT Services panel.

- To start Administration Server from Netscape Console:

  1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).

  2. In the Console tab, locate the Administration Server instance that you want to start, and double-click the corresponding entry.

     The Administration Server window appears.

  3. In the Tasks tab, click Start the Server.

- To start Administration Server from the command line:

  At the prompt, enter the following line:

  `<server_root>/admin-<instance_id>/start-admin`

  `<server_root>` is the directory where the Administration Server binaries are kept. You first specified this directory during server installation.

  `<instance_id>` is the ID for this instance of Administration Server. You first specified this during server installation.

  This command starts Administration Server at the port number you specified during installation. Once the server is running, you can use Netscape Console to access Certificate Management System.

- Administration Server runs as a service in a Windows NT system. You can use the Windows NT Services panel to start the service directly.

Shutting Down Administration Server

It is good security practice to shut down Administration Server when you are not using it. This minimizes the chances of someone else changing your configuration.

You can shut down the server from Netscape Console, the command line, or the Windows NT Services panel.

- To shut down Administration Server from Netscape Console:

  1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).

  2. In the Console tab, locate the Administration Server instance that you want to shut down, and double-click the corresponding entry.

     The Administration Server window appears.

  3. In the Tasks tab, click Stop the Server.

- To shut down Administration Server from the command line:

  At the prompt, enter the following line:

  ```
  <server_root>/admin-<instance_id>/stop-admin
  ```

  `<server_root>` is the directory where the Administration Server binaries are kept. You first specified this directory during server installation.

  `<instance_id>` is the ID for this instance of Administration Server. You first specified this during server installation.

- Administration Server runs as a service in a Windows NT system; you can use the Windows NT Services panel to stop the service directly.

Logging In to Netscape Console

You can launch and use Netscape Console only when the configuration directory and Administration Server are running. If the servers are not running, go to the command line and start them. For information on starting Administration Server from the command line, see "Starting Administration Server" on page 67. For information on starting the configuration directory, check the Netscape Directory Server documentation.

When you launch Netscape Console, it displays a login window. You are required to authenticate to the configuration directory by entering your administrator's ID, your password, and the URL (including port number) of the

Administration Server representing a server group to which you have access. You cannot use Netscape Console without having login access to at least one server group on your network.

1. Open the Netscape Console application by using the appropriate option:

   - Local access in Unix

     At the command-line prompt, enter the following line:

     `<server_root>/admin-<instance_id>/start-console`

     `<server_root>` is the directory where the Administration Server binaries are kept. You first specified this directory during server installation.

     `<instance_id>` is the ID for this instance of Administration Server. You first specified this during server installation.

   - Local access in Windows NT

     Double-click the Netscape Console icon on your desktop; this icon was created when you installed your first Netscape version 4.x server.

     The Netscape Console window appears.

2. Authenticate yourself to the configuration directory.

   User ID. Type the *administrator ID* you specified when you installed Administration Server on your machine. You installed Administration Server either when you installed your first Netscape 4.x server or as a part of CMS installation.

   Password. Type the *administrator* password that you specified when you installed Administration Server on your computer during CMS installation.

   Administration URL. This field should show the URL to Administration Server. If it doesn't or if it doesn't have the URL of Administration Server that you want, type the URL in this field. The URL is based on the computer host name and the Administration Server port number you chose when you installed Certificate Management System. Use this format:

   `http://<machine_name>.<your_domain>.<domain>:<port_number>`

For example, if your domain name is `siroe` and you installed Administration Server on a host machine called `myHost` and specified port number `12345`, the URL would look like this:

```
http://myHost.siroe.com:12345
```

3. Click OK.

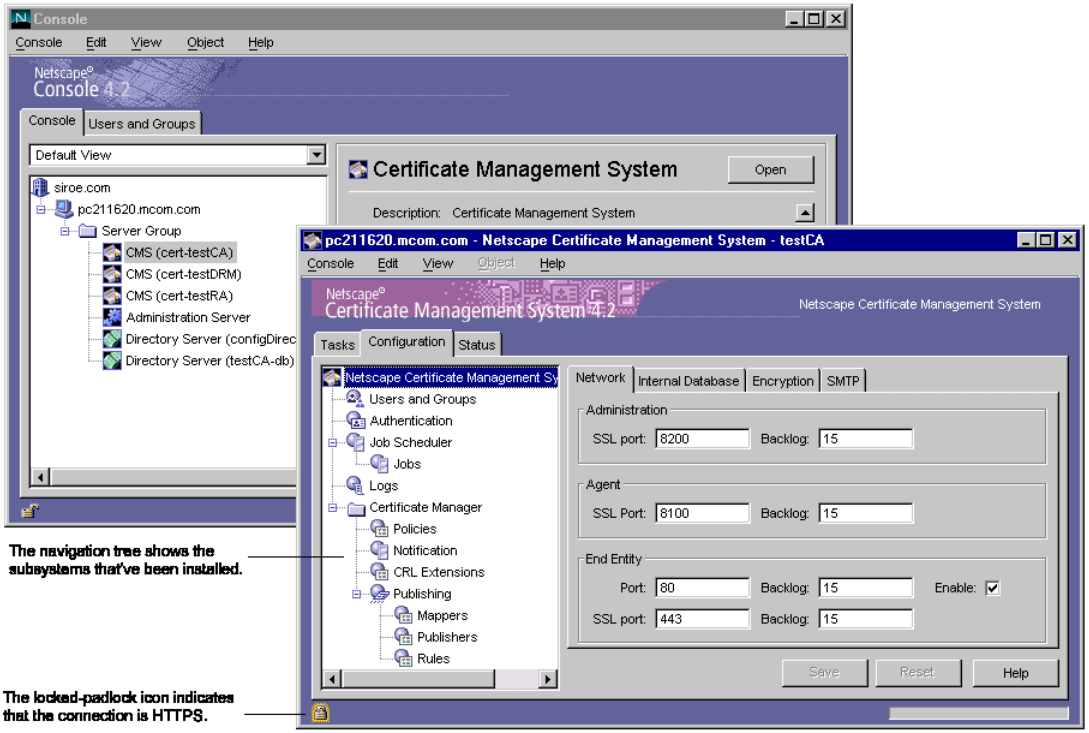
   Netscape Console appears with a list of all the servers and resources under your control (see Figure 2.1).

   - To view general information about a specific server, click the corresponding entry.
   - To access the administration interface for a specific server, double-click the corresponding entry.

The CMS Window

The CMS window is a GUI-based administration interface that allows you to perform day-to-day operational and managerial duties for Certificate Management System. You launch the CMS window from within Netscape Console (see Figure 2.3). You can use the CMS window to access the server locally or remotely.

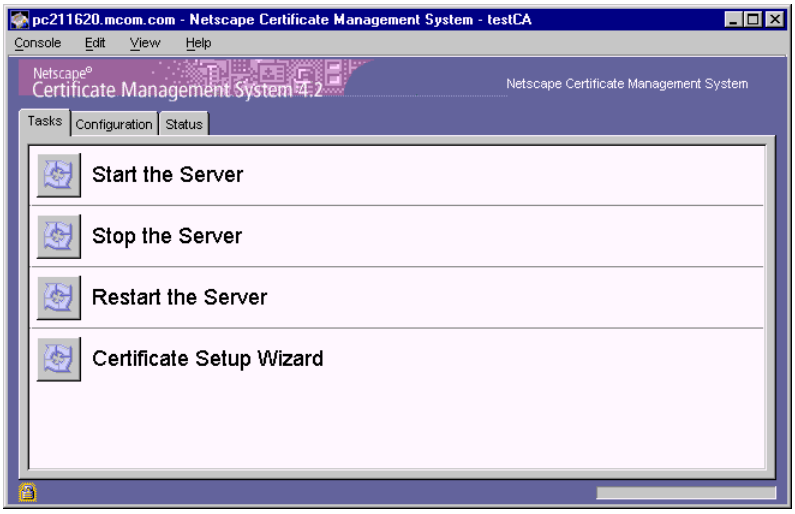
Figure 2.3 Certificate Management System window, launched from Netscape Console



The CMS window has three separate tabs, each addressing specific administrative areas: the Tasks tab, the Configuration tab, and the Status tab.

Tasks Tab

The Tasks tab allows you to perform frequently required tasks. From this tab, you can start, stop, and restart the server.



For details on these common tasks, see “Stopping Certificate Management System” on page 137 and “Restarting Certificate Management System” on page 139.

Configuration Tab

The Configuration tab allows you to view the current server configuration settings and change them.

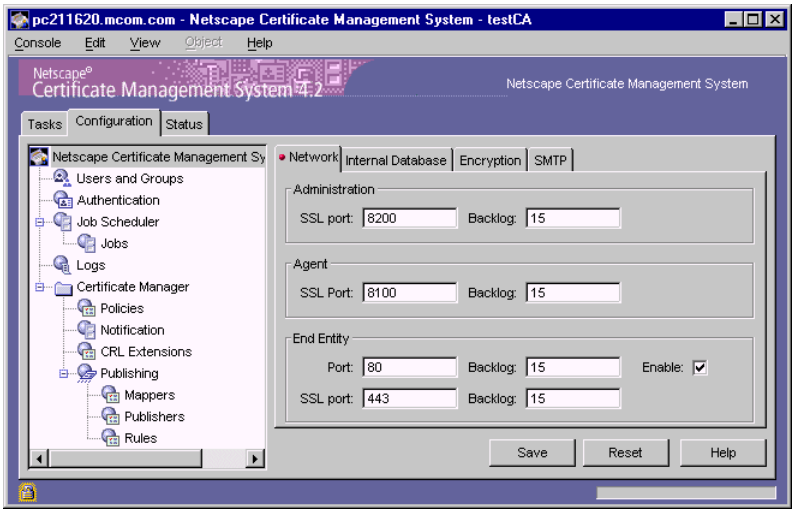


Table 2.1 provides details about the tasks you can accomplish from this tab. You access specific settings by selecting an entry in the navigation tree and working with the tabs that appear in the right pane.

Table 2.1 Tasks you can accomplish from the Configuration tab

Task	Description
Configuring network settings	This involves operations such as the following: • Changing the administration, agent, and end-entity ports of Certificate Management System • Changing the names of CMS instances For details, see “Configuring Ports, Database, and SMTP Settings” on page 155.
Configuring the internal database settings	This involves specifying the host name and port number of the directory server that Certificate Management System should use for storing data. For details, see “Internal Database” on page 163.

Table 2.1 Tasks you can accomplish from the Configuration tab (Continued)

Task	Description
Managing CMS keys and certificates	This involves operations such as the following: • Managing the CMS certificate database. For details, see "Managing the Certificate Database" on page 294. • Generating new and renewing existing certificates for the Certificate Manager, Registration Manager, and Data Recovery Manager. For details, see "Getting New Certificates for the Subsystems" on page 277. • Installing new hardware tokens. For details, see "Tokens for Storing Keys and Certificates" on page 235. • Configuring the Data Recovery Manager to archive and recover end users' encryption private keys. For details, see "Recovering Encrypted Data" on page 1115.
Configuring SMTP settings	This involves specifying the host name and port number of the mail server that Certificate Management System should use for sending email notifications. For details, see "SMTP Settings" on page 168.
Setting up privileged users	This involves operations such as the following: • Entering information about privileged users (administrators, agents, and trusted managers) into the CMS internal database. For details, see "Setting Up Privileged Users" on page 190. • Modifying user information. For details, see "Changing Privileged-User Information" on page 219. • Deleting users from the database. For details, see "Deleting a Privileged User" on page 223.
Determining authentication for end users	This involves operations such as the following: • Viewing currently registered authentication plug-in modules. For details, see "Configuring Authentication for End Users" on page 383. • Configuring Certificate Management System to use a specific authentication method to authenticate end entities when they request service from the server. For details, see "Configuring Authentication for End Users" on page 383. • Registering custom authentication plug-in modules. For details, see "Developing Custom Authentication Modules" on page 417.

Table 2.1 Tasks you can accomplish from the Configuration tab (Continued)

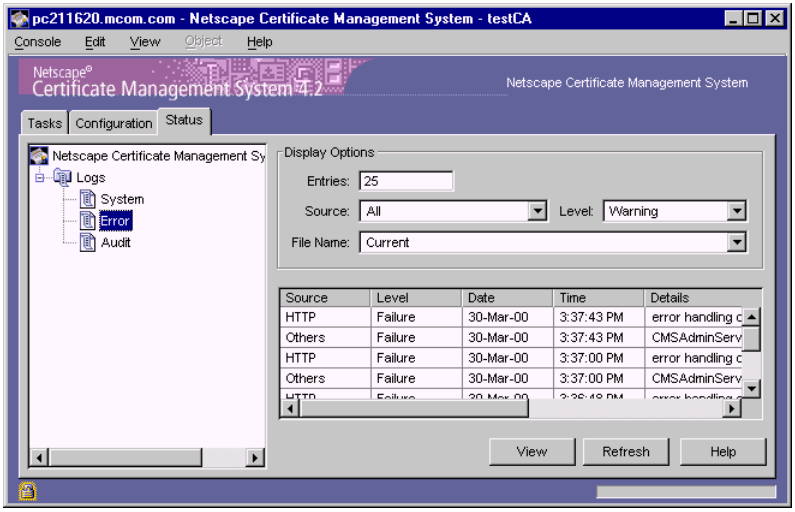
Task	Description
Scheduling jobs	This involves operations such as the following: • Viewing currently registered plug-in modules for jobs. For details, see "Configuring Schedulable Jobs" on page 467. • Configuring Certificate Management System to execute specific jobs. For details, see "Configuring Schedulable Jobs" on page 467.
Enabling automated notifications	This involves operations such as the following: • Entering the information required by the server to send automated notifications to one or more agents when a request enters the agent queue. For details, see "Notification of New Request in Queue" on page 453. • Entering the information required by the server to send automated certificate-issuance notifications to end entities when the server issues them certificates. For details, see "Notifications of Certificate Issuance to End Entities" on page 450. • Customizing the notification message templates to suit your organization's requirements. For details, see "Customizing Message Templates" on page 459.
Configuring certificate issuance and management policies	This involves operations such as the following: • Viewing currently registered policy plug-in modules for a Certificate Manager or Registration Manager. • Configuring the Certificate Manager or Registration Manager for certificate formulation, issuance, renewal, and revocation policies, and configuring the Data Recovery Manager for the archiving and recovery of end users' encryption private keys. For details, see "Setting up Policy Rules for a Subsystem" on page 689.
Publishing certificates and CRLs	This involves operations such as the following: • Configuring the Certificate Manager to publish certificates and CRLs to an LDAP-compliant directory, such as Netscape Directory Server. For details, see "Publishing Certificates and CRLs to a Directory" on page 786. • Configuring the Certificate Manager to publish certificates and CRLs to a flat file for importing into other repositories. For details, see "Publishing Certificates and CRLs to Flat Files" on page 840. • Configuring the Certificate Manager to publish CRLs to an online validation authority, such as ValiCert Certificate VA. For details, see "Publishing CRLs to Online Validation Authority" on page 857.

Table 2.1 Tasks you can accomplish from the Configuration tab (Continued)

Task	Description
Managing CMS logs	This involves configuring system, error, and audit logs maintained by Certificate Management System. For details, see "Configuring Logs" on page 1058.
Configuring the Data Recovery Manager	This involves configuring the Data Recovery Manager for archival and recovery of end users' encryption private keys. For details, see "Recovering Encrypted Data" on page 1115.
Backing up and restoring CMS data	This involves operations such as the following: • Periodically backing up the CMS data. • In the event of data loss, using the resulting archives to restore the data. For details, see "Backing Up and Restoring Data" on page 1173.

Status Tab

The Status tab allows you to monitor the server by viewing the contents of various logs maintained by Certificate Management System.



You can monitor active as well as rotated log files. For details, see the following sections:

- "Monitoring System Logs" on page 1065
- "Monitoring Error Logs" on page 1068
- "Monitoring Audit Logs" on page 1070

Logging In to the CMS Window

You access the CMS window from Netscape Console. For details on Netscape Console, see "Netscape Console" on page 64.

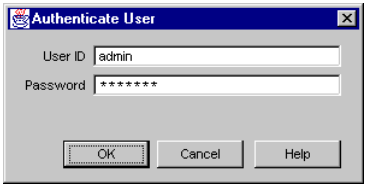
The Console tab of Netscape Console contains a list of network resources that are under your control. In this list you can identify CMS instances by their icons or by server identifiers you specified during installation (for example, you may have named a CMS instance ABC Corp CA).

To open the CMS window for a specific CMS instance:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. In the Console tab, select the Server Group that contains the CMS instance you want to use as your source.
3. In the navigation tree, locate the CMS instance you want to administer.
4. Select the instance and click Open or double-click the corresponding entry.

   If the selected server is not running, you are asked to start the server first. In that case, start the server, and then repeat steps 2 through 4. For information on starting the server, see "Starting Certificate Management System" on page 128.

If the selected server is running, you are prompted to authenticate to Certificate Management System.



5. Enter the appropriate information:

   User ID. If you are logging in for the first time, type the `Certificate Administrator ID`; you specified this user ID during installation (so that you could log in to the CMS window without having to create privileged-user entries). Otherwise, type your privileged-user ID (administrator ID).

   Password. If you are logging in for the first time, type the `Certificate Administrator` password; you specified this password during installation (so that you could log in to the CMS window without having to create privileged-user entries). Otherwise, type your privileged-user (administrator) password; see "Administrators" on page 172.

   Upon successful authentication, the CMS window appears (Figure 2.3 on page 72).

Note Accessing the CMS window is a privileged operation that is restricted to CMS administrators. After you log in for the first time, create at least one user in each of the default groups; see "Groups and Their Privileges" on page 186.

Chapter

3

Configuration

The runtime properties of Netscape Certificate Management System (CMS) are governed by a set of configuration parameters. These parameters are stored in a file that is read by the server during startup.

When you install Certificate Management System, the installer creates an ASCII file, named `CMS.cfg`, and populates it with the appropriate configuration parameters. You can control the way Certificate Management System functions by making the appropriate changes to the configuration information.

This chapter explains how the installation affects the number of configuration files created in your machine and their contents. It also explains ways in which you can modify the configuration and precautions you should take when doing so. The chapter ends with a road map to configuring individual subsystems.

The chapter has the following sections:

Effects of Installation Type on Configuration

For each instance of Certificate Management System there is a configuration file, named `CMF.cfg`. The configuration file controls the runtime properties of the corresponding CMS instance.

A CMS instance can include a single subsystem or two subsystems in one of the following combinations:

- A single Certificate Manager, Registration Manager, or Data Recovery Manager
- A Certificate Manager and Data Recovery Manager together
- A Registration Manager and Data Recovery Manager together

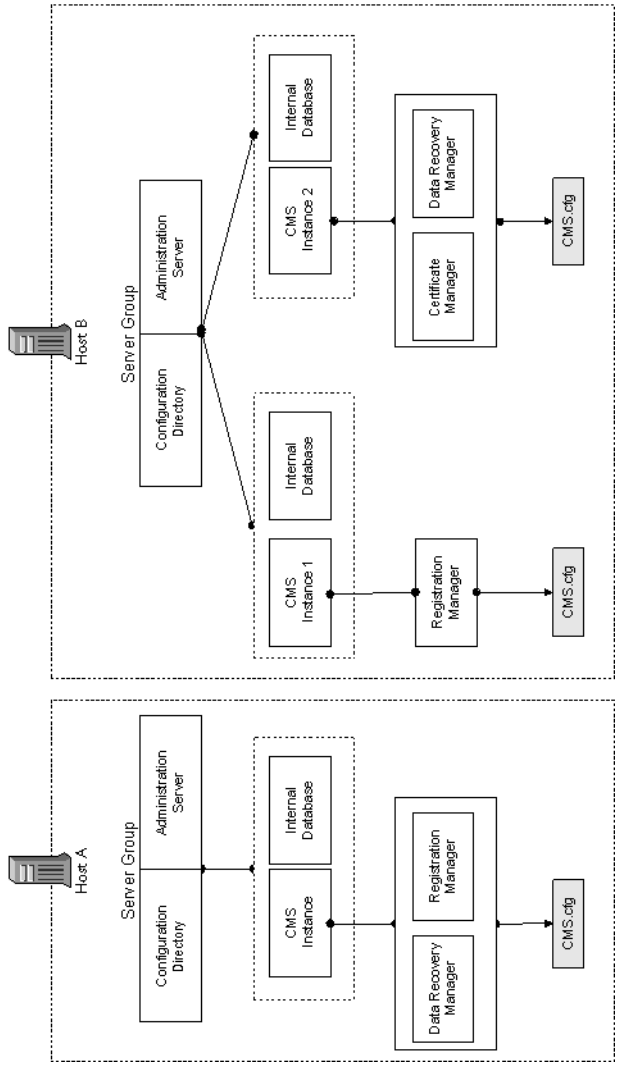
Figure 3.1 on page 83 illustrates a deployment scenario involving two instances of Certificate Management System running on the same host (Host A) and a single instance running on another host (Host B). Notice the two separate configuration files for the instances running on Host A, one for each CMS instance.

Although the names of both the configuration files are the same, the information included in the files differs according to the subsystems installed in each instance. For example, the configuration file for *CMS Instance 1* includes only those parameters that govern the Registration Manager, whereas the configuration file for *CMS Instance 2* includes parameters that control both the Certificate Manager and Data Recovery Manager.

It is also important to understand that subsystems installed in a CMS instance share certain parts of the configuration. They use the same

- Administration, agent, and end-entity ports for interaction
- Internal token and trust database
- SSL ciphers during SSL negotiation
- Privileged users (administrators and agents)
- Log files to log messages
- Internal database for data storage

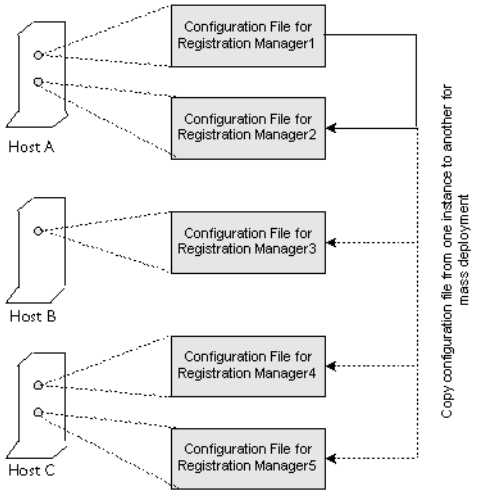
Figure 3.1 How installation affects configuration



Duplicating a Configuration from One Instance to Another

If you have deployed a large number of CMS instances that are identical—for example, multiple Registration Managers—and you want all these instances to have the same configuration, you can accomplish this by configuring one of the instances and then replacing the configuration files of the other instances with the one that contains the required configuration. Figure 3.2 illustrates this quick way of deploying multiple Registration Managers with the same configuration.

Figure 3.2 Duplicating a configuration



Caution Be careful when replacing configuration of one instance with another. The configuration file for an instance contains instance-specific parameters. If you replace these parameters, the instance will fail to start or function properly.

Locating the Configuration File

Each instance of Certificate Management System has its own configuration file, `CMF.cfg`. The default location for this file is as follows:

`<server_root>/cert-<instance_id>/config`

`<server_root>` is the directory where the CMS binaries are kept. You first specified this directory during installation.

`<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.

Modifying the Configuration

You can modify the CMS configuration in two ways:

- By changing the configuration parameter values from the CMS window. This is the recommended method for changing configuration. See "Changing the Configuration From the CMS Window" on page 85.
- By manually changing the configuration parameter values in the configuration file, `CMF.cfg`. See "Changing the Configuration by Editing the Configuration File" on page 86.

Changing the Configuration From the CMS Window

The CMS window allows you to view the current configuration of a CMS instance and make the required changes. Because this is the recommended method for changing configuration, the chapters that follow focus on explaining how to change the various configuration parameter values from the CMS window.

Note You may find the road map provided in "Road Map to Configuring Subsystems" on page 105 useful in setting up your CMS instances.

Changing the Configuration by Editing the Configuration File

This section explains how to change the CMS configuration by editing the configuration parameter values in the file `CMF.cfg`. This ASCII file is read by Certificate Management System when it is started.

Caution Do not edit the configuration file directly if you are not familiar with the configuration parameters or if you are not sure that the changes you intend to make are acceptable by the server. Certificate Management System will fail to start up if you make incorrect modifications to the configuration file. Incorrect configuration can also result in data loss.

Also, before you start editing the configuration file, be sure to read "Guidelines for Editing the Configuration File" on page 87.

To modify the configuration file directly:

1. Stop the CMS instance whose configuration file you want to edit (see "Stopping Certificate Management System" on page 137).
2. Open a terminal window.
3. Go to this directory: `<server_root>/cert-<instance_id>/config`
4. Open the configuration file, `CMF.cfg`, in a text editor.
5. Edit information in the file and save your changes.
6. Restart Certificate Management System (see "Restarting Certificate Management System" on page 139).

Guidelines for Editing the Configuration File

The file-based, configuration-store implementation for Certificate Management System is based on `java.util.Properties`. The following guidelines may help you interpret the information in the configuration file.

- The format of the configuration file is as follows:

  ```
  #comment
  [parameter]=value
  ```

```
value

[parameter]
multi
line
value (e.g. base-64 encoded certificate)
```

- Comment lines, blank lines, unknown parameters, or misspelled parameters are ignored by Certificate Management System. Comment lines begin with a number sign (#). A line beginning with white space is considered a continuation of the previous line.
- The configuration file has many sections. Some sections contain parameters specific to the subsystems that have been installed; the other sections contain parameters that are shared by the subsystems. Subsystem-specific parameters are distinguished by a prefix identifying the subsystem:
  - `ca` for the Certificate Manager
  - `ra` for the Registration Manager
  - `kra` for the Data Recovery Manager
- The parameter names and their values are strings. The parameter names can be hierarchically structured with `'.'` notation with multiple levels—for example, `ca.Policy.rule.RSAKeyRule.maxSize`. The entries corresponding to a lower level (such as `Policy` in the example) can be requested from the configuration corresponding to its higher level (`ca` in the example).
- The values that need to be localized (such as distinguished names in multibyte format) should be entered in `utf8` format. For more information on this format, see the document *UTF-8, a transformation format of Unicode and ISO 10646*, available at this URL:

  `http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2044.txt`
- Certificate Management System writes out the configuration in a sorted order.

- The values of some parameters are referenced to other parts of the configuration file. For example, assume that a parameter is defined as `subsystem.id=ca`; when this parameter is processed by the server, all the parameters beginning with `ca` will be used.
- The configuration file supports Unix-style file separator, the forward slash (/). If the \ file separator is required, use two backward slashes (\\) instead of one.

Sample Configuration File

The following sample configuration is of a Certificate Manager.

Important This sample file includes some of the parameters used by Certificate Management System. However, there is no guarantee that an arbitrary set of options you create will work.

```
_000=##
_001=## File Created On     : Sun Jan 02 23:02:35 PST 2000
_002=##

instanceRoot=/usr/netscape/cert-testCA
machineName=testCA.siroe.com

agentGateway._000=##
agentGateway._001=## Agent Gateway
agentGateway._002=##

   agentGateway.docRoot=/usr/netscape/cert-testCA/web/agent
   agentGateway.dynamicVariables=serverdate=serverdate()
   agentGateway.enableAdminEnroll=true
   agentGateway.enableBulkInterface=true
   agentGateway.keepAliveOn=true
   agentGateway.mimeTypeConf=/usr/netscape/cert-testCA/config/
     mime.types
   agentGateway.numServices=1
   agentGateway.service0=https
   agentGateway.CAGetBySerial.successTemplate=/ca/ImportCert.template
   agentGateway.adminEnroll.successTemplate=/ca/EnrollSuccess.template
   agentGateway.bulkissuance.errorTemplate=/ca/bulkissuance.template
   agentGateway.bulkissuance.pendingTemplate=/ca/bulkissuance.template
   agentGateway.bulkissuance.rejectedTemplate=/ca/bulkissuance.template
```

```
    agentGateway.bulkissuance.successTemplate=/ca/bulkissuance.template
    agentGateway.bulkissuance.svcpendingTemplate=/ca/
      bulkissuance.template
    agentGateway.bulkissuance.unauthorizedTemplate=/ca/
      bulkissuance.template
    agentGateway.bulkissuance.unexpectedErrorTemplate=/ca/
      bulkissuance.template
    agentGateway.https.backlog=15
    agentGateway.https.nickName=Server-Cert cert-testCA
    agentGateway.https.port=4605
    agentGateway.https.type=https

auths._000=##
auths._001=## Authentication
auths._002=##
auths.impl._000=##
auths.impl._001=## authentication manager implementations
auths.impl._002=##

    auths.impl.KerberosAuth.class=com.netscape.certsrv.
      authentication.KerberosBasedAuthentication
    auths.impl.NISAuth.class=com.netscape.certsrv.
      authentication.NISAuth
    auths.impl.PortalEnroll.class=com.netscape.certsrv.
      authentication.PortalEnroll
    auths.impl.UidPwdDirAuth.class=com.netscape.certsrv.
      authentication.UidPwdDirAuthentication
    auths.impl.UidPwdPinDirAuth.class=com.netscape.certsrv.
    authentication.UidPwdPinDirAuthentication
    auths.revocationChecking.bufferSize=5
    auths.revocationChecking.ca=ca
    auths.revocationChecking.enabled=true
    auths.revocationChecking.unknownStateInterval=0
    auths.revocationChecking.validityInterval=120

ca.id=ca
    ca.local=true

    ca.Policy.order=KeyAlgRule, RSAKeyRule, DefaultValidityRule,
      RenewalConstraintsRule, DefaultRenewalValidityRule,
      RevocationConstraintsRule, DefaultRevocationRule, NSCertTypeExt,
      CMCertKeyUsageExt, RMCertKeyUsageExt, ClientCertKeyUsageExt,
      ServerCertKeyUsageExt, ObjSignCertKeyUsageExt,
      SubjectKeyIdentifierExt, CertificatePoliciesExt, NSCComment,
```

```
      OCSPNoCheckExt, OCSPSigningExt, CODESigningExt, GenericASN1Ext,
      CRLDistributionPointsExt, SubjectAltNameExt, SigningAlgRule,
      AuthorityKeyIdentifierExt, BasicConstraintsExt, UniqueSubjectName,
      NameConstraintsExt, PolicyConstraintsExt, SubCANameCheck,
      PolicyMappingsExt, IssuerRule

ca.Policy.impl._000=##
ca.Policy.impl._001=## Policy Implementations
ca.Policy.impl._002=##

   ca.Policy.impl.AuthInfoAccessExt.class=com.netscape.certsrv.policy.
     AuthInfoAccessExt
   ca.Policy.impl.AuthorityKeyIdentifierExt.class=com.netscape.certsrv.
     policy.AuthorityKeyIdentifierExt
   ca.Policy.impl.BasicConstraintsExt.class=com.netscape.certsrv.
     policy.BasicConstraintsExt
   ca.Policy.impl.CRLDistributionPointsExt.class=com.netscape.certsrv.
     policy.CRLDistributionPointsExt
   ca.Policy.impl.CertificatePoliciesExt.class=com.netscape.certsrv.
     policy.CertificatePoliciesExt
   ca.Policy.impl.DSAKeyConstraints.class=com.netscape.certsrv.policy.
     DSAKeyConstraints
   ca.Policy.impl.DefaultRevocation.class=com.netscape.certsrv.policy.
     DefaultRevocation
   ca.Policy.impl.ExtendedKeyUsageExt.class=com.netscape.certsrv.
     policy.ExtendedKeyUsageExt
   ca.Policy.impl.GenericASN1Ext.class=com.netscape.certsrv.policy.
     GenericASN1Ext
   ca.Policy.impl.IssuerAltNameExt.class=com.netscape.certsrv.policy.
     IssuerAltNameExt
   ca.Policy.impl.IssuerConstraints.class=com.netscape.certsrv.policy.
     IssuerConstraints
   ca.Policy.impl.KeyAlgorithmConstraints.class=com.netscape.certsrv.
     policy.KeyAlgorithmConstraints
   ca.Policy.impl.KeyUsageExt.class=com.netscape.certsrv.policy.
     KeyUsageExt
   ca.Policy.impl.NSCComment.class=com.netscape.certsrv.policy.
     NSCComment
   ca.Policy.impl.NSCertTypeExt.class=com.netscape.certsrv.policy.
     NSCertTypeExt
   ca.Policy.impl.NameConstraintsExt.class=com.netscape.certsrv.policy.
     NameConstraintsExt
   ca.Policy.impl.OCSPNoCheckExt.class=com.netscape.certsrv.policy.
     OCSPNoCheckExt
   ca.Policy.impl.PinPresent.class=com.netscape.certsrv.policy.
     PinPresent
```

```
ca.Policy.impl.PolicyConstraintsExt.class=com.netscape.certsrv.
  policy.PolicyConstraintsExt
ca.Policy.impl.PolicyMappingsExt.class=com.netscape.certsrv.policy.
  PolicyMappingsExt
ca.Policy.impl.PrivateKeyUsagePeriodExt.class=com.netscape.certsrv.
  policy.PrivateKeyUsagePeriodExt
ca.Policy.impl.RSAKeyConstraints.class=com.netscape.certsrv.policy.
  RSAKeyConstraints
ca.Policy.impl.RenewalConstraints.class=com.netscape.certsrv.policy.
  RenewalConstraints
ca.Policy.impl.RenewalValidityConstraints.class=com.netscape.
  certsrv.policy.RenewalValidityConstraints
ca.Policy.impl.RevocationConstraints.class=com.netscape.certsrv.
  policy.RevocationConstraints
ca.Policy.impl.SigningAlgorithmConstraints.class=com.netscape.
  certsrv.policy.SigningAlgorithmConstraints
ca.Policy.impl.SubCANameCheck.class=com.netscape.certsrv.policy.
  SubCANameCheck
ca.Policy.impl.SubjectAltNameExt.class=com.netscape.certsrv.policy.
  SubjAltNameExt
ca.Policy.impl.SubjectDirectoryAttributesExt.class=com.netscape.
  certsrv.policy.SubjectDirectoryAttributesExt
ca.Policy.impl.SubjectKeyIdentifierExt.class=com.netscape.certsrv.
  policy.SubjectKeyIdentifierExt
ca.Policy.impl.UniqueSubjectName.class=com.netscape.certsrv.policy.
  UniqueSubjectName
ca.Policy.impl.ValidityConstraints.class=com.netscape.certsrv.
  policy.ValidityConstraints

ca.Policy.rule.AuthorityKeyIdentifierExt.enable=true
ca.Policy.rule.AuthorityKeyIdentifierExt.implName=
  AuthorityKeyIdentifierExt
ca.Policy.rule.AuthorityKeyIdentifierExt.predicate=

ca.Policy.rule.BasicConstraintsExt.enable=true
ca.Policy.rule.BasicConstraintsExt.implName=BasicConstraintsExt
ca.Policy.rule.BasicConstraintsExt.predicate=
ca.Policy.rule.BasicConstraintsExt.removeBasicExt=true

ca.Policy.rule.CMCertKeyUsageExt.crlSign=true
ca.Policy.rule.CMCertKeyUsageExt.digitalSignature=true
ca.Policy.rule.CMCertKeyUsageExt.enable=true
ca.Policy.rule.CMCertKeyUsageExt.implName=KeyUsageExt
ca.Policy.rule.CMCertKeyUsageExt.keyCertsign=true
```

```
ca.Policy.rule.CMCertKeyUsageExt.nonRepudiation=true
ca.Policy.rule.CMCertKeyUsageExt.predicate=certType==ca

ca.Policy.rule.CODESigningExt.critical=false
ca.Policy.rule.CODESigningExt.enable=true
ca.Policy.rule.CODESigningExt.id0=1.3.6.1.5.5.7.3.3
ca.Policy.rule.CODESigningExt.implName=ExtendedKeyUsageExt
ca.Policy.rule.CODESigningExt.predicate=certType==codeSignClient

ca.Policy.rule.CRLDistributionPointsExt.enable=false
ca.Policy.rule.CRLDistributionPointsExt.implName=
CRLDistributionPointsExt
ca.Policy.rule.CRLDistributionPointsExt.issuerName0=
ca.Policy.rule.CRLDistributionPointsExt.issuerName1=
ca.Policy.rule.CRLDistributionPointsExt.issuerName2=
ca.Policy.rule.CRLDistributionPointsExt.issuerType0=
ca.Policy.rule.CRLDistributionPointsExt.issuerType1=
ca.Policy.rule.CRLDistributionPointsExt.issuerType2=
ca.Policy.rule.CRLDistributionPointsExt.numPoints=0
ca.Policy.rule.CRLDistributionPointsExt.pointName0=
ca.Policy.rule.CRLDistributionPointsExt.pointName1=
ca.Policy.rule.CRLDistributionPointsExt.pointName2=
ca.Policy.rule.CRLDistributionPointsExt.pointType0=
ca.Policy.rule.CRLDistributionPointsExt.pointType1=
ca.Policy.rule.CRLDistributionPointsExt.pointType2=
ca.Policy.rule.CRLDistributionPointsExt.predicate=
ca.Policy.rule.CRLDistributionPointsExt.reasons0=
ca.Policy.rule.CRLDistributionPointsExt.reasons1=
ca.Policy.rule.CRLDistributionPointsExt.reasons2=

ca.Policy.rule.CertificatePoliciesExt.enable=false
ca.Policy.rule.CertificatePoliciesExt.implName=
  CertificatePoliciesExt
ca.Policy.rule.CertificatePoliciesExt.policyId=
ca.Policy.rule.CertificatePoliciesExt.predicate=

ca.Policy.rule.ClientCertKeyUsageExt.digitalSignature=true
ca.Policy.rule.ClientCertKeyUsageExt.enable=true
ca.Policy.rule.ClientCertKeyUsageExt.implName=KeyUsageExt
ca.Policy.rule.ClientCertKeyUsageExt.keyEncipherment=true
ca.Policy.rule.ClientCertKeyUsageExt.nonRepudiation=true
ca.Policy.rule.ClientCertKeyUsageExt.predicate=certType==client
```

```
ca.Policy.rule.DSAKeyRule.enable=true
ca.Policy.rule.DSAKeyRule.implName=DSAKeyConstraints
ca.Policy.rule.DSAKeyRule.maxSize=2048
ca.Policy.rule.DSAKeyRule.minSize=512
ca.Policy.rule.DSAKeyRule.predicate=

ca.Policy.rule.DefaultRenewalValidityRule.enable=true
ca.Policy.rule.DefaultRenewalValidityRule.implName=
  RenewalValidityConstraints
ca.Policy.rule.DefaultRenewalValidityRule.maxValidity=365
ca.Policy.rule.DefaultRenewalValidityRule.minValidity=30
ca.Policy.rule.DefaultRenewalValidityRule.predicate=
ca.Policy.rule.DefaultRenewalValidityRule.renewalInterval=15

ca.Policy.rule.DefaultRevocationRule.enable=true
ca.Policy.rule.DefaultRevocationRule.implName=DefaultRevocation
ca.Policy.rule.DefaultRevocationRule.predicate=

ca.Policy.rule.DefaultValidityRule.enable=true
ca.Policy.rule.DefaultValidityRule.implName=ValidityConstraints
ca.Policy.rule.DefaultValidityRule.maxValidity=365
ca.Policy.rule.DefaultValidityRule.minValidity=30
ca.Policy.rule.DefaultValidityRule.predicate=

ca.Policy.rule.GenericASN1Ext.critical=false
ca.Policy.rule.GenericASN1Ext.enable=false
ca.Policy.rule.GenericASN1Ext.implName=GenericASN1Ext
ca.Policy.rule.GenericASN1Ext.name=
ca.Policy.rule.GenericASN1Ext.oid=
ca.Policy.rule.GenericASN1Ext.pattern=
ca.Policy.rule.GenericASN1Ext.predicate=
ca.Policy.rule.GenericASN1Ext.attribute.0.source=
ca.Policy.rule.GenericASN1Ext.attribute.0.type=
ca.Policy.rule.GenericASN1Ext.attribute.0.value=
ca.Policy.rule.GenericASN1Ext.attribute.1.source=
ca.Policy.rule.GenericASN1Ext.attribute.1.type=
ca.Policy.rule.GenericASN1Ext.attribute.1.value=
ca.Policy.rule.GenericASN1Ext.attribute.2.source=
ca.Policy.rule.GenericASN1Ext.attribute.2.type=
ca.Policy.rule.GenericASN1Ext.attribute.2.value=
ca.Policy.rule.GenericASN1Ext.attribute.3.source=
```

```
ca.Policy.rule.GenericASN1Ext.attribute.3.type=
ca.Policy.rule.GenericASN1Ext.attribute.3.value=
ca.Policy.rule.GenericASN1Ext.attribute.4.source=
ca.Policy.rule.GenericASN1Ext.attribute.4.type=
ca.Policy.rule.GenericASN1Ext.attribute.4.value=
ca.Policy.rule.GenericASN1Ext.attribute.5.source=
ca.Policy.rule.GenericASN1Ext.attribute.5.type=
ca.Policy.rule.GenericASN1Ext.attribute.5.value=
ca.Policy.rule.GenericASN1Ext.attribute.6.source=
ca.Policy.rule.GenericASN1Ext.attribute.6.type=
ca.Policy.rule.GenericASN1Ext.attribute.6.value=
ca.Policy.rule.GenericASN1Ext.attribute.7.source=
ca.Policy.rule.GenericASN1Ext.attribute.7.type=
ca.Policy.rule.GenericASN1Ext.attribute.7.value=
ca.Policy.rule.GenericASN1Ext.attribute.8.source=
ca.Policy.rule.GenericASN1Ext.attribute.8.type=
ca.Policy.rule.GenericASN1Ext.attribute.8.value=
ca.Policy.rule.GenericASN1Ext.attribute.9.source=
ca.Policy.rule.GenericASN1Ext.attribute.9.type=
ca.Policy.rule.GenericASN1Ext.attribute.9.value=

ca.Policy.rule.IssuerRule.enable=false
ca.Policy.rule.IssuerRule.implName=IssuerConstraints
ca.Policy.rule.IssuerRule.issuerDN=
ca.Policy.rule.IssuerRule.predicate=certType==client AND
  certauthEnroll==on

ca.Policy.rule.KeyAlgRule.algorithms=RSA
ca.Policy.rule.KeyAlgRule.enable=true
ca.Policy.rule.KeyAlgRule.implName=KeyAlgorithmConstraints
ca.Policy.rule.KeyAlgRule.predicate=

ca.Policy.rule.NSCComment.enable=false
ca.Policy.rule.NSCComment.implName=NSCComment
ca.Policy.rule.NSCComment.policyId=
ca.Policy.rule.NSCComment.predicate=

ca.Policy.rule.NSCertTypeExt.enable=true
ca.Policy.rule.NSCertTypeExt.implName=NSCertTypeExt
ca.Policy.rule.NSCertTypeExt.predicate=certType!=CEP-Request

ca.Policy.rule.NameConstraintsExt.critical=true
```

```
ca.Policy.rule.NameConstraintsExt.enable=false
ca.Policy.rule.NameConstraintsExt.implName=NameConstraintsExt
ca.Policy.rule.NameConstraintsExt.numExcludedSubtrees=3
ca.Policy.rule.NameConstraintsExt.numPermittedSubtrees=3
ca.Policy.rule.NameConstraintsExt.predicate=certType == ca
ca.Policy.rule.NameConstraintsExt.excludedSubtrees0.base=
ca.Policy.rule.NameConstraintsExt.excludedSubtrees0.max=-1
ca.Policy.rule.NameConstraintsExt.excludedSubtrees0.min=0
ca.Policy.rule.NameConstraintsExt.excludedSubtrees0.valueType=
ca.Policy.rule.NameConstraintsExt.excludedSubtrees1.base=
ca.Policy.rule.NameConstraintsExt.excludedSubtrees1.max=-1
ca.Policy.rule.NameConstraintsExt.excludedSubtrees1.min=0
ca.Policy.rule.NameConstraintsExt.excludedSubtrees1.valueType=
ca.Policy.rule.NameConstraintsExt.excludedSubtrees2.base=
ca.Policy.rule.NameConstraintsExt.excludedSubtrees2.max=-1
ca.Policy.rule.NameConstraintsExt.excludedSubtrees2.min=0
ca.Policy.rule.NameConstraintsExt.excludedSubtrees2.valueType=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees0.base=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees0.max=-1
ca.Policy.rule.NameConstraintsExt.permittedSubtrees0.min=0
ca.Policy.rule.NameConstraintsExt.permittedSubtrees0.valueType=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees1.base=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees1.max=-1
ca.Policy.rule.NameConstraintsExt.permittedSubtrees1.min=0
ca.Policy.rule.NameConstraintsExt.permittedSubtrees1.valueType=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees2.base=
ca.Policy.rule.NameConstraintsExt.permittedSubtrees2.max=-1
ca.Policy.rule.NameConstraintsExt.permittedSubtrees2.min=0
ca.Policy.rule.NameConstraintsExt.permittedSubtrees2.valueType=

ca.Policy.rule.OCSPNoCheckExt.critical=false
ca.Policy.rule.OCSPNoCheckExt.enable=true
ca.Policy.rule.OCSPNoCheckExt.implName=OCSPNoCheckExt
ca.Policy.rule.OCSPNoCheckExt.predicate=certType==ocspResponder

ca.Policy.rule.OCSPSigningExt.critical=false
ca.Policy.rule.OCSPSigningExt.enable=true
ca.Policy.rule.OCSPSigningExt.id0=1.3.6.1.5.5.7.3.9
ca.Policy.rule.OCSPSigningExt.implName=ExtendedKeyUsageExt
ca.Policy.rule.OCSPSigningExt.predicate=certType==ocspResponder

ca.Policy.rule.ObjSignCertKeyUsageExt.digitalSignature=true
```

```
ca.Policy.rule.ObjSignCertKeyUsageExt.enable=true
ca.Policy.rule.ObjSignCertKeyUsageExt.implName=KeyUsageExt
ca.Policy.rule.ObjSignCertKeyUsageExt.keyCertsign=true
ca.Policy.rule.ObjSignCertKeyUsageExt.predicate=
  certType==objSignClient

ca.Policy.rule.PolicyConstraintsExt.critical=false
ca.Policy.rule.PolicyConstraintsExt.enable=false
ca.Policy.rule.PolicyConstraintsExt.implName=PolicyConstraintsExt
ca.Policy.rule.PolicyConstraintsExt.inhibitPolicyMapping=0
ca.Policy.rule.PolicyConstraintsExt.predicate=certType==ca
ca.Policy.rule.PolicyConstraintsExt.reqExplicitPolicy=0

ca.Policy.rule.PolicyMappingsExt.critical=false
ca.Policy.rule.PolicyMappingsExt.enable=false
ca.Policy.rule.PolicyMappingsExt.implName=PolicyMappingsExt
ca.Policy.rule.PolicyMappingsExt.numPolicyMappings=1
ca.Policy.rule.PolicyMappingsExt.predicate=certType==ca
ca.Policy.rule.PolicyMappingsExt.policyMap0.issuerDomainPolicy=
ca.Policy.rule.PolicyMappingsExt.policyMap0.subjectDomainPolicy=

ca.Policy.rule.RMCertKeyUsageExt.digitalSignature=true
ca.Policy.rule.RMCertKeyUsageExt.enable=true
ca.Policy.rule.RMCertKeyUsageExt.implName=KeyUsageExt
ca.Policy.rule.RMCertKeyUsageExt.nonRepudiation=true
ca.Policy.rule.RMCertKeyUsageExt.predicate=certType==ra

ca.Policy.rule.RSAKeyRule.enable=false
ca.Policy.rule.RSAKeyRule.exponents=3,7,17,65537
ca.Policy.rule.RSAKeyRule.implName=RSAKeyConstraints
ca.Policy.rule.RSAKeyRule.maxSize=2048
ca.Policy.rule.RSAKeyRule.minSize=512
ca.Policy.rule.RSAKeyRule.predicate=

ca.Policy.rule.RenewalConstraintsRule.enable=true
ca.Policy.rule.RenewalConstraintsRule.implName=RenewalConstraints
ca.Policy.rule.RenewalConstraintsRule.predicate=

ca.Policy.rule.RevocationConstraintsRule.enable=true
ca.Policy.rule.RevocationConstraintsRule.implName=
  RevocationConstraints
ca.Policy.rule.RevocationConstraintsRule.predicate=
```

```
    ca.Policy.rule.ServerCertKeyUsageExt.dataEncipherment=true
    ca.Policy.rule.ServerCertKeyUsageExt.digitalSignature=true
    ca.Policy.rule.ServerCertKeyUsageExt.enable=true
    ca.Policy.rule.ServerCertKeyUsageExt.implName=KeyUsageExt
    ca.Policy.rule.ServerCertKeyUsageExt.keyEncipherment=true
    ca.Policy.rule.ServerCertKeyUsageExt.nonRepudiation=true
    ca.Policy.rule.ServerCertKeyUsageExt.predicate=certType==server

    ca.Policy.rule.SigningAlgRule.algorithms=
      MD5withRSA,MD2withRSA,SHA1withRSA,SHA1withDSA
    ca.Policy.rule.SigningAlgRule.enable=true
    ca.Policy.rule.SigningAlgRule.implName=SigningAlgorithmConstraints
    ca.Policy.rule.SigningAlgRule.predicate=

    ca.Policy.rule.SubCANameCheck.enable=true
    ca.Policy.rule.SubCANameCheck.implName=SubCANameCheck
    ca.Policy.rule.SubCANameCheck.predicate=
    ca.Policy.rule.SubjectAltNameExt.enable=true
    ca.Policy.rule.SubjectAltNameExt.enableManualValues=false
    ca.Policy.rule.SubjectAltNameExt.implName=SubjectAltNameExt

    ca.Policy.rule.SubjectKeyIdentifierExt.enable=true
    ca.Policy.rule.SubjectKeyIdentifierExt.implName=
      SubjectKeyIdentifierExt
    ca.Policy.rule.SubjectKeyIdentifierExt.predicate=certType==ca

    ca.Policy.rule.UniqueSubjectName.enable=false
    ca.Policy.rule.UniqueSubjectName.implName=UniqueSubjectName
    ca.Policy.rule.UniqueSubjectName.predicate=

ca.crl._000=##
ca.crl._001=## CA CRL
ca.crl._002=##

    ca.crl.MasterCRL.allowExtensions=false
    ca.crl.MasterCRL.autoUpdateInterval=20
    ca.crl.MasterCRL.class=com.netscape.certsrv.ca.CRLIssuingPoint
    ca.crl.MasterCRL.description=
      CA's complete Certificate Revocation List

    ca.notification.certIssued.emailSubject=Your Certificate Request
```

```
ca.notification.certIssued.emailTemplate=/usr/netscape/cert-testCA/
  emails/certIssued_CA.html
ca.notification.certIssued.enabled=false
ca.notification.certIssued.senderEmail=

ca.notification.requestInQ.emailSubject=Certificate Request in Queue
ca.notification.requestInQ.emailTemplate=/usr/netscape/cert-testCA/
  emails/reqInQueue.html
ca.notification.requestInQ.enabled=false
ca.notification.requestInQ.recipientEmail=
ca.notification.requestInQ.senderEmail=

ca.publish.mapper.impl.LdapDNCompsMap.class=com.netscape.certsrv.
  ldap.LdapCertCompsMap
ca.publish.mapper.impl.LdapDNExactMap.class=com.netscape.certsrv.
  ldap.LdapCertExactMap
ca.publish.mapper.impl.LdapSimpleMap.class=com.netscape.certsrv.
  ldap.LdapSimpleMap
ca.publish.mapper.impl.LdapSubjAttrMap.class=com.netscape.certsrv.
  ldap.LdapCertSubjMap
ca.publish.mapper.instance.LdapCaCertMap.dnPattern=
  UID=$cert.cn,OU=people,O=$cert.o
ca.publish.mapper.instance.LdapCaCertMap.pluginName=LdapSimpleMap
ca.publish.mapper.instance.LdapCrlMap.dnPattern=
  UID=$cert.cn,OU=people,O=$cert.o
ca.publish.mapper.instance.LdapCrlMap.pluginName=LdapSimpleMap
ca.publish.mapper.instance.LdapUserCertMap.dnPattern=
  UID=$cert.UID,OU=people,O=$cert.o
ca.publish.mapper.instance.LdapUserCertMap.pluginName=LdapSimpleMap
ca.publish.publisher.impl.FileBasedPublisher.class=com.netscape.
  certsrv.ldap.FileBasedPublisher
ca.publish.publisher.impl.LdapCaCertPublisher.class=com.netscape.
  certsrv.ldap.LdapCaCertPublisher
ca.publish.publisher.impl.LdapCrlPublisher.class=com.netscape.
  certsrv.ldap.LdapCrlPublisher
ca.publish.publisher.impl.LdapUserCertPublisher.class=com.netscape.
  certsrv.ldap.LdapUserCertPublisher
ca.publish.publisher.impl.ValiCertPublisher.class=com.valicert.
  publisher.VcPublisher

ca.publish.publisher.instance.LdapCaCertPublisher.caCertAttr=
  caCertificate;binary
ca.publish.publisher.instance.LdapCaCertPublisher.caObjectClass=
  certificationAuthority
ca.publish.publisher.instance.LdapCaCertPublisher.pluginName=
```

```
  LdapCaCertPublisher
ca.publish.publisher.instance.LdapCrlPublisher.crlAttr=
  certificateRevocationList;binary
ca.publish.publisher.instance.LdapCrlPublisher.pluginName=
  LdapCrlPublisher
ca.publish.publisher.instance.LdapUserCertPublisher.certAttr=
  userCertificate;binary
ca.publish.publisher.instance.LdapUserCertPublisher.pluginName=
  LdapUserCertPublisher
ca.publish.rule.impl.Rule.class=com.netscape.certsrv.ldap.LdapRule

ca.publish.rule.instance.LdapCaCertRule.enable=true
ca.publish.rule.instance.LdapCaCertRule.mapper=LdapCaCertMap
ca.publish.rule.instance.LdapCaCertRule.pluginName=Rule
ca.publish.rule.instance.LdapCaCertRule.predicate=
ca.publish.rule.instance.LdapCaCertRule.publisher=
  LdapCaCertPublisher
ca.publish.rule.instance.LdapCaCertRule.type=ca

ca.publish.rule.instance.LdapCrlRule.enable=true
ca.publish.rule.instance.LdapCrlRule.mapper=LdapCrlMap
ca.publish.rule.instance.LdapCrlRule.pluginName=Rule
ca.publish.rule.instance.LdapCrlRule.predicate=
ca.publish.rule.instance.LdapCrlRule.publisher=LdapCrlPublisher
ca.publish.rule.instance.LdapCrlRule.type=crl

ca.publish.rule.instance.LdapObjSignCertRule.enable=true
ca.publish.rule.instance.LdapObjSignCertRule.mapper=LdapUserCertMap
ca.publish.rule.instance.LdapObjSignCertRule.pluginName=Rule
ca.publish.rule.instance.LdapObjSignCertRule.predicate=
ca.publish.rule.instance.LdapObjSignCertRule.publisher=
  LdapUserCertPublisher
ca.publish.rule.instance.LdapObjSignCertRule.type=objSignClient

ca.publish.rule.instance.LdapUserCertRule.enable=true
ca.publish.rule.instance.LdapUserCertRule.mapper=LdapUserCertMap
ca.publish.rule.instance.LdapUserCertRule.pluginName=Rule
ca.publish.rule.instance.LdapUserCertRule.predicate=
ca.publish.rule.instance.LdapUserCertRule.publisher=
  LdapUserCertPublisher
ca.publish.rule.instance.LdapUserCertRule.type=client

ca.signing.cacertnickname=caSigningCert cert-testCA
```

```
    ca.signing.defaultSigningAlgorithm=MD5withRSA
    ca.signing.tokenname=Internal Key Storage Token

    dbs.ldap=internaldb
    dbs.newSchemaEntryAdded=true
    dbs.nextSerialNumber=1

eeGateway._000=##
eeGateway._001=## End Entity Gateway
eeGateway._002=##

    eeGateway.authority=ca
    eeGateway.docRoot=/usr/netscape/cert-testCA/web/ee
    eeGateway.dynamicVariables=serverdate=serverdate(),subsystemname=
      subsystemname(),http=http()
    eeGateway.enableConnector=true
    eeGateway.keepAliveOn=true
    eeGateway.mimeTypeConf=/usr/netscape/cert-testCA/config/mime.types
    eeGateway.numServices=2
    eeGateway.service0=http
    eeGateway.service1=https
    eeGateway.http.backlog=15
    eeGateway.http.enable=true
    eeGateway.http.port=4603
    eeGateway.http.type=http
    eeGateway.https.backlog=15
    eeGateway.https.nickName=Server-Cert cert-testCA
    eeGateway.https.port=4604
    eeGateway.https.type=https

internaldb._000=##
internaldb._001=## Internal Database
internaldb._002=##
    internaldb.maxConns=15
    internaldb.minConns=3
    internaldb.ldapauth.authtype=BasicAuth
    internaldb.ldapauth.bindDN=cn=Directory Manager
    internaldb.ldapauth.bindPWPrompt=Internal LDAP Database
    internaldb.ldapconn.host=testCA.siroe.com
    internaldb.ldapconn.port=3602
    internaldb.ldapconn.secureConn=false
```

```
jobsScheduler._000=##
jobsScheduler._001=## jobScheduler
jobsScheduler._002=##
    jobsScheduler.enabled=false
    jobsScheduler.interval=1
    jobsScheduler.impl.RenewalNotificationJob.class=com.netscape.
      certsrv.jobs.RenewalNotificationJob
    jobsScheduler.impl.RequestInQueueJob.class=com.netscape.
      certsrv.jobs.RequestInQueueJob
    jobsScheduler.impl.UnpublishExpiredJob.class=com.netscape.
      certsrv.jobs.UnpublishExpiredJob

    jobsScheduler.job.certRenewalNotifier.cron=0 3 * * 1-5
    jobsScheduler.job.certRenewalNotifier.emailSubject=
      Certificate Renewal Notification
    jobsScheduler.job.certRenewalNotifier.emailTemplate=/usr/netscape/
      cert-testCA/emails/rnJob1.txt
    jobsScheduler.job.certRenewalNotifier.enabled=false
    jobsScheduler.job.certRenewalNotifier.notifyEndOffset=30
    jobsScheduler.job.certRenewalNotifier.notifyTriggerOffset=30
    jobsScheduler.job.certRenewalNotifier.pluginName=
      RenewalNotificationJob
    jobsScheduler.job.certRenewalNotifier.senderEmail=
    jobsScheduler.job.certRenewalNotifier.summary.emailSubject=
      Certificate Renewal Notification Summary
    jobsScheduler.job.certRenewalNotifier.summary.emailTemplate=/usr/
      netscape/cert-testCA/emails/rnJob1Summary.txt
    jobsScheduler.job.certRenewalNotifier.summary.enabled=true
    jobsScheduler.job.certRenewalNotifier.summary.itemTemplate=/usr/
      netscape/cert-testCA/emails/rnJob1Item.txt
    jobsScheduler.job.certRenewalNotifier.summary.recipientEmail=
    jobsScheduler.job.certRenewalNotifier.summary.senderEmail=

    jobsScheduler.job.requestInQueueNotifier.cron=0 0 * * 0
    jobsScheduler.job.requestInQueueNotifier.enabled=false
    jobsScheduler.job.requestInQueueNotifier.pluginName=
      RequestInQueueJob
    jobsScheduler.job.requestInQueueNotifier.subsystemId=ca
    jobsScheduler.job.requestInQueueNotifier.summary.emailSubject=
      Requests in Queue Summary Report
    jobsScheduler.job.requestInQueueNotifier.summary.emailTemplate=/usr/
      netscape/cert-testCA/emails/riq1Summary.html
    jobsScheduler.job.requestInQueueNotifier.summary.enabled=true
    jobsScheduler.job.requestInQueueNotifier.summary.recipientEmail=
```

```
    jobsScheduler.job.requestInQueueNotifier.summary.senderEmail=

    jobsScheduler.job.unpublishExpiredCerts.cron=0 0 * * 6
    jobsScheduler.job.unpublishExpiredCerts.enabled=false
    jobsScheduler.job.unpublishExpiredCerts.pluginName=
      UnpublishExpiredJob
    jobsScheduler.job.unpublishExpiredCerts.summary.emailSubject=
      Expired Certs Unpublished Summary
    jobsScheduler.job.unpublishExpiredCerts.summary.emailTemplate=/usr/
      netscape/cert-testCA/emails/euJob1.html
    jobsScheduler.job.unpublishExpiredCerts.summary.enabled=true
    jobsScheduler.job.unpublishExpiredCerts.summary.itemTemplate=/usr/
      netscape/cert-testCA/emails/euJob1Item.html
    jobsScheduler.job.unpublishExpiredCerts.summary.recipientEmail=
    jobsScheduler.job.unpublishExpiredCerts.summary.senderEmail=

jss._000=##
jss._001=## JSS
jss._002=##
    jss.certdb=/usr/netscape/cert-testCA/config/cert7.db
    jss.enable=true
    jss.keydb=/usr/netscape/cert-testCA/config/key3.db
    jss.moddb=/usr/netscape/admin-serv/config/secmodule.db
    jss.ssl.cipherfortezza=true
    jss.ssl.cipherpref=
    jss.ssl.cipherversion=cipherdomestic

logAudit._000=##
logAudit._001=## Logging
logAudit._002=##
    logAudit.bufferSize=512
    logAudit.expirationTime=2592000
    logAudit.fileName=/usr/netscape/cert-testCA/logs/audit
    logAudit.flushInterval=5
    logAudit.level=1
    logAudit.maxFileSize=100
    logAudit.on=true
    logAudit.rolloverInterval=2592000

logError._000=##
logError._001=## Logging
logError._002=##
```

```
    logError.bufferSize=512
    logError.expirationTime=2592000
    logError.fileName=/usr/netscape/cert-testCA/logs/error
    logError.flushInterval=5
    logError.level=3
    logError.maxFileSize=100
    logError.on=true
    logError.rolloverInterval=2592000

    logNTAudit.NTEventSourceName=cert-testCA
    logNTAudit.level=1
    logNTAudit.on=true
    logNTSystem.NTEventSourceName=cert-testCA
    logNTSystem.level=2
    logNTSystem.on=true

logSystem._000=##
logSystem._001=## Logging
logSystem._002=##
    logSystem.bufferSize=512
    logSystem.expirationTime=2592000
    logSystem.fileName=/usr/netscape/cert-testCA/logs/system
    logSystem.flushInterval=5
    logSystem.level=3
    logSystem.maxFileSize=100
    logSystem.on=true
    logSystem.rolloverInterval=2592000

    oidmap.auth_info_access.class=com.netscape.certsrv.cert.
      AuthInfoAccessExtension
    oidmap.auth_info_access.oid=1.3.6.1.5.5.7.1.1
    oidmap.challenge_password.class=com.netscape.certsrv.cmsgateway.
      cert.crs.ChallengePassword
    oidmap.challenge_password.oid=1.2.840.113549.1.9.7
    oidmap.extended_key_usage.class=com.netscape.certsrv.cert.
      ExtendedKeyUsageExtension
    oidmap.extended_key_usage.oid=2.5.29.37
    oidmap.extensions_requested_pkcs9.class=com.netscape.certsrv.
      cmsgateway.cert.crs.ExtensionsRequested
    oidmap.extensions_requested_pkcs9.oid=1.2.840.113549.1.9.14
    oidmap.extensions_requested_vsgn.class=com.netscape.certsrv.
      cmsgateway.cert.crs.ExtensionsRequested
    oidmap.extensions_requested_vsgn.oid=2.16.840.1.113733.1.9.8
```

```
    oidmap.netscape_comment.class=netscape.security.x509.
      NSCCommentExtension
    oidmap.netscape_comment.oid=2.16.840.1.113730.1.13
    oidmap.ocsp_no_check.class=com.netscape.certsrv.cert.
      OCSPNoCheckExtension
    oidmap.ocsp_no_check.oid=1.3.6.1.5.5.7.48.1.5
    os.serverName=cert-testCA
    os.userid=nobody

radm._000=##
radm._001=## Remote Admin
radm._002=##
    radm.keepAliveOn=true
    radm.mimeTypeConf=/usr/netscape/cert-testCA/config/mime.types
    radm.numServices=1
    radm.service0=https
    radm.https.backlog=15
    radm.https.maxThreads=10
    radm.https.minThreads=3
    radm.https.nickName=Server-Cert cert-testCA
    radm.https.port=4606
    radm.https.timeout=0
    radm.https.type=https

    smtp.host=localhost
    smtp.port=25

subsystem._000=##
subsystem._001=## Loadable Subsystems
subsystem._002=##
    subsystem.0.class=com.netscape.certsrv.ca.CertificateAuthority
    subsystem.0.id=ca
    subsystem.1.class=com.netscape.certsrv.cmsgateway.EEGateway
    subsystem.1.id=eeGateway

usrgrp._000=##
usrgrp._001=## User/Group
usrgrp._002=##
    usrgrp.ldap=internaldb
```

Road Map to Configuring Subsystems

This section outlines how to configure an instance of Certificate Management System and indicates where to find the information required to accomplish the task.

Step 1. Check Which Subsystems are Installed in the Instance

Log in to the CMS window for the CMS instance you installed, and check the navigation tree to see which subsystems are installed in that instance; this way you will know the subsystems you should configure. To log in to the CMS window, see "Logging In to the CMS Window" on page 78.

Step 2. Check the Port Numbers

Check the port numbers assigned for administration, agent, and end-entity operations. Make the appropriate modifications, if necessary. Keep in mind that all subsystems installed in an instance use the same ports, but can be configured to listen on different IP addresses. For instructions, see "Configuring Port Numbers" on page 158.

Step 3. Verify Key Pair and Certificates

When you install a CMS instance, the server prompts you to create the certificates required for the subsystems in that instance to function. You should check the certificates used by each subsystem, and determine if you need to get additional certificates, use hardware tokens, and so on.

- Each subsystem in an instance has a set of certificates that it uses for specific purposes. Understand how and when the subsystem uses its certificates. For details, see "Keys and Certificates for the Main Subsystems" on page 226.
- Determine if you want to generate any new certificates. For example, if you have two subsystems installed in an instance, you may want them to use separate SSL server certificates; by default, there's only one SSL server certificate per instance. For details, see "Getting New Certificates for the Subsystems" on page 277.
- Determine if you want to use hardware tokens for generating and storing these certificates. If required, install new hardware tokens. For details, see "Tokens for Storing Keys and Certificates" on page 235.

- Determine if you want to renew any of the existing certificates. For example, if you have issued certificates with very short validity periods, you might want to renew them. For details, see "Renewing Certificates for the Subsystems" on page 286.

- Check the certificate database to see which CA certificates are trusted. Delete any unwanted CA certificates, change the trust settings of CA certificates that you don't want to trust to *untrusted*, and install any new CA certificate or certificate chains. For details, see "Managing the Certificate Database" on page 294.

Step 4. Check the SMTP Settings

Check the mail server settings—Certificate Management System uses this information to send automated email notifications. If necessary, make the appropriate changes to the host name and port number. Keep in mind that all subsystems installed in an instance use the same mail server. To change the mail server-specific information, see "SMTP Settings" on page 168.

Step 5. Set up Privileged Users

Set up required administrators and agents. This way you can delegate administration and agent tasks to other individuals. For details, see "Setting Up Privileged Users" on page 190.

If you have installed remote Registration Managers that have certificates signed by third-party CAs (that is, not by a Certificate Manager), you should add their certificates to the Certificate Manager's database to facilitate SSL client authenticated communication. For details, see "Setting Up Trusted Managers" on page 201.

Step 6. Customize End-Entity and Agent Forms

End entities can interact with the Certificate Manager and Registration Manager with the help of end-entity forms; end entities cannot interact with the Data Recovery Manager. Similarly, agents can interact with the appropriate subsystem using the agent forms. Certificate Management System provides HTML forms-based interfaces for end entities and agents out of the box. For details, see "Introduction to End-Entity and Agent Interfaces" on page 895.

Determine which forms you want to use for end-entity enrollment and whether they require any customization. You may also use your own forms for this purpose, provided you add the required JavaScript. For details, see "End-Entity Services" on page 895.

When customizing end-entity forms, keep in mind the authentication method—manual or automated—you want to employ for your end entities.

Step 7. Setup Authentication for End Entities

Depending on how you've deployed Certificate Management System, you may need to do this for a Certificate Manager or Registration Manager, or for both. For example, you may have a PKI setup in which Registration Managers act as front ends to Certificate Managers—that is, end entities interact with Registration Managers only; they do not interact with the Certificate Manager.

1. Determine whether you want to use any of the authentication plug-in modules provided for automated enrollment of end users. For details, see "Overview of Authentication Modules" on page 320. If you don't, you either have to use the manual enrollment method or will have to use your own custom module. For information on developing authentication modules and registering them in the CMS framework, see "Developing Custom Authentication Modules" on page 417.

2. Configure the Certificate Manager or Registration Manager to use a specific authentication method. For details, see "Setting Up Authentication for End-User Enrollment" on page 387.

Step 8. Schedule Jobs

Each CMS instance includes a *Job Scheduler* component that can execute specific jobs at specified times. The Job Scheduler functions similar to a traditional Unix *cron* daemon in that it takes registered cron jobs and executes them at a preconfigured date and time. For details, see "Introduction to Job Scheduling and Notifications" on page 433.

During installation, a few jobs are already created and enabled. Jobs that you might want to schedule include email notifications of timed events (such as the expiration of a certificate) that require action on the part of users, and periodic activities such as removing expired certificates from the publishing directory. For scheduling jobs, follow the instructions in "Scheduling Automated Jobs" on page 471.

Step 9: Enable Event-Driven Notifications

You can also configure both Certificate Manager and Registration Manager to send email notifications automatically to end entities, agents, or administrators when certain events occur. Unlike jobs that are executed at preconfigured schedule, these notifications are event-driven—that is, whenever an event occurs, the server notifies the user. Notifiable events include certificate issuance and pending requests in an agent queue.

Decide if you want to turn on any of the notifications. If you do, the server uses the mail server specified in the SMTP settings (see "Step 4. Check the SMTP Settings" on page 106) to send these notifications. For details, see "Event-Driven Notifications" on page 449.

Step 10. Set up Policies

Each subsystem in a CMS instance has its own policy processor. If you have installed more than one subsystem in an instance, you should apply the instructions in this section to each subsystem. That is, you should configure the Certificate Manager and Registration Manager for certificate formulation, issuance, renewal, and revocation policies. Similarly, configure the Data Recovery Manager for key archival and recovery policies. To understand policy, see "Introduction to Policy" on page 487.

1. During installation, a few policy rules are already created and enabled. Check each policy rule and decide whether you want to use it. If you don't, you can either disable it or delete it altogether from the configuration. For those rules that you want to use, check the configuration parameter values and make changes as appropriate.

2. Determine if you want to add any new policy rules. Check the built-in policy plug-in modules to see if they can be used to create the rules you want. You can also plug-in your own modules in the CMS framework and use them.

3. Add new rules, if required.

For instructions to do all of the above tasks, see "Setting up Policy Rules for a Subsystem" on page 689.

Step 11. Set up Publishing

This step is optional, and is applicable to the Certificate Manager only—you need to do this only if you want the Certificate Manager to publish certificates and CRLs to any of the supported repositories.

- To configure a Certificate Manager to publish certificates and CRLs to an LDAP-compliant directory, such as Netscape Directory Server, see "Publishing Certificates and CRLs to a Directory" on page 786.
- To configure a Certificate Manager to publish certificates and CRLs to a flat file, see "Publishing Certificates and CRLs to Flat Files" on page 840.
- To configure a Certificate Manager to publish CRLs to an online validation authority (also called OCSP responder), see "Publishing CRLs to Online Validation Authority" on page 857.

Step 12. Set up Logging

Each instance of Certificate Management System maintains extensive audit, error, and system logs. By looking at these logs, you can monitor a server's activities. Also, by configuring these logs, you can control the information that gets written to the log files. Because Certificate Management System maintains the log files in the file system of the host machine, it is important that you configure the logs appropriately (so that the host machine doesn't get overloaded). Be sure to read "Introduction to Logs" on page 1045; this chapter will help you decide log configuration.

Once you decide the configuration for server logs, follow the information in "Configuring Logs" on page 1058 and configure all the three logs. Then, start monitoring the server's activities as explained in "Monitoring Logs" on page 1064.

Step 13. Set up archival and recovery for end users' keys

If you have installed the Data Recovery Manager, follow the instructions in "Setting Up Key Archival and Recovery Process" on page 1133 and set up archival and recovery for end users' encryption private keys.

Step 14. Test your PKI Setup

Use the information in "Issuing and Managing End-Entity Certificates" on page 1081 and test that the certificate issuance, renewal, and revocation operations work satisfactorily.

If you have deployed the Data Recovery Manager, follow the information in "Step A. Test Your Key Archival Setup" on page 1145 and "Step D. Test Your Key Recovery Setup" to test the key archival and recovery operation respectively.

Step 15. Plan for Backing up CMS Configuration and Data

It is a good practice to periodically back up the CMS data on to some backup media. Creating backups will help you use them for data restoration in the event of data loss. For details, see "Backing Up and Restoring Data" on page 1173.

2

Managing Certificate Management System

Chapter

4

Installing and Uninstalling CMS Instances

After the initial installation of Netscape Certificate Management System (CMS), you may need to install additional instances, remove unwanted instances, or duplicate configuration in multiple instances. This chapter describes how to manage these tasks by using Netscape Console, the single, unified administration interface for your network.

You can use Netscape Console only when Netscape Administration Server is running. During CMS installation, you specified a port number for the Administration Server instance you will use to administer Certificate Management System. If Administration Server is shut down, be sure to start it at this port. To minimize security risks, shut down the Administration Server when you have finished using Netscape Console. For more information about Netscape Console, see "Administration Tasks and Tool" on page 63.

The chapter has the following sections:

Installing Multiple Instances

Multiple instances of Certificate Management System can run on the same machine. You might, for example, install multiple Registration Managers, all reporting to the same Certificate Manager, to handle requests from different types of users (end users, servers, and routers) or from users from different domains. For example deployment scenarios, see the *Netscape Certificate Management System Installation and Deployment Guide*.

Once Certificate Management System is installed on a machine, you can use that CMS installation to create multiple instances of the server on the same machine. Administration Server contains all the files necessary to install another instance of Certificate Management System on the same machine; you don't have to run the complete installation (`setup`) program again. However, you do need to run the CMS installation wizard each time you create an instance, so that you can configure the server and generate the required certificates. So, before attempting to create another instance of Certificate Management System, be sure to read about the installation wizard in the *Netscape Certificate Management System Installation and Deployment Guide*.

When you install additional CMS instances on the same machine, you are required to specify different ports for each CMS instance to listen on. For example, you will have to set up one server to listen on port 443 and another server to listen on port 4430. However, if you install multiple CMS instances on a machine that has been set up with more than one IP addresses, you can configure each instance to listen to a specific IP address—this enables you to use same the port numbers for different CMS instances installed on the same machine.

Keep in mind that when you create a new instance, you'll be required to specify different port numbers; the installation wizard allows you to specify the port numbers only, not IP addresses. After you have successfully created the instance, you can edit the CMS configuration file to specify the IP addresses for individual CMS ports and then change the port numbers. For details on editing the configuration file to include the IP addresses, see "Step 2: Specify IP Addresses" on page 161. For details on changing the port numbers, see "Configuring Port Numbers" on page 158.

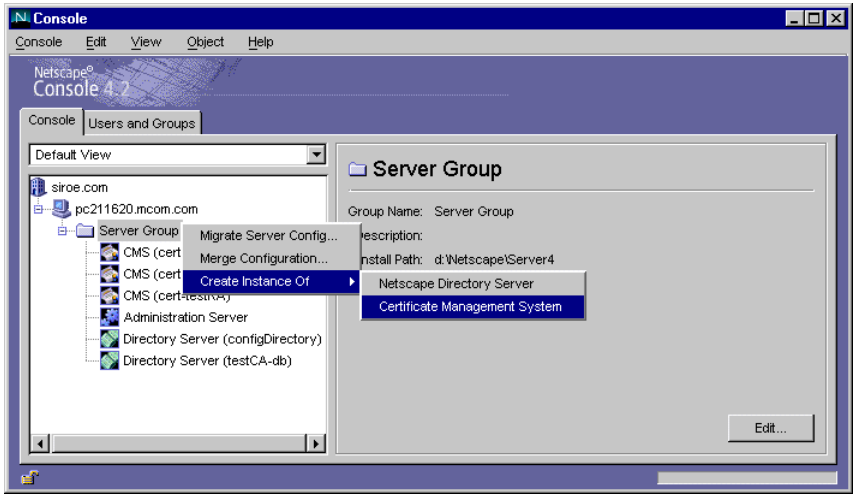
To create another instance of Certificate Management System with a separate configuration file (and certificates):

1. Log in to Netscape Console (see “Logging In to Netscape Console” on page 69).

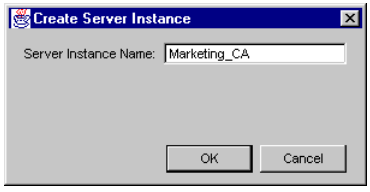
2. In the Console tab, select the server group that contains the CMS instance you want to use as your source.

3. From the Object menu, choose the Create Instance Of option and, in the pop-up menu that appears, choose Certificate Management System.

   As shown in this figure, you can also right-click to choose this option from the pop-up menu.



   The Create Server Instance dialog box appears.



4. Type a name for the instance. For the name, you can use any combination of letters (`aA` to `zZ`), digits (`0` to `9`), an underscore (_), and a hyphen (`-`); other characters and spaces are not allowed. For example, you can type `Netscape_root-CA` as the instance name, but not `Netscape root CA`.

5. Click OK.

   The instance you created appears in the navigation tree. Note that the instance name appears in this form: `CMS` (`cert-<instance_id>`)

   `where <instance_id>` is the name you specified for the new CMS instance.

   For example, if you named the instance `Marketing_CA`, the instance name in the navigation tree will be `CMS` (`cert-Marketing_CA`).

6. To start the installation wizard, double-click the new instance in the navigation tree.

   Use the installation wizard to finish configuring the new instance.

Viewing Instance Information

In Netscape Console, you can view some of the basic information—the name and version number of the server, the directory in which it's installed, and date it was installed—about a CMS instance.

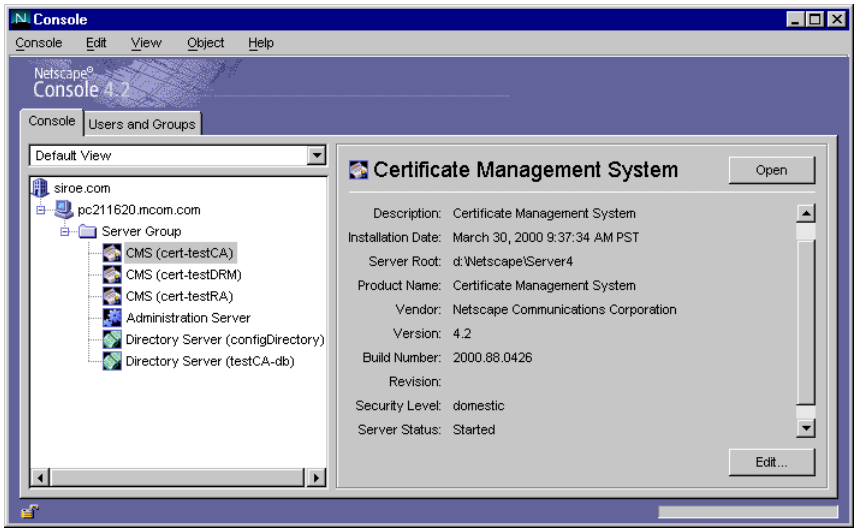
To view information pertaining to a specific CMS instance:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).

2. In the Console tab, double-click the server group that contains the CMS instance you want to view.

3. In the list of server instances, select the CMS instance you want to view.

The right pane shows information about the selected CMS instance.



The information displayed includes the following:

Server Name. A descriptive name of the CMS instance. You can change this name; see “Changing the Name of an Instance” on page 120).

Description. Additional information that helps you identify the CMS instance. You can change this description; see “Changing the Name of an Instance” on page 120.

Installation Date. The date the server was installed.

Server Root. The directory that holds all the files for the selected CMS instance, the files of its Administration Server, and the files of any other Netscape servers in the same server group (that is, administered by that Administration Server). A host typically has only one server root, but more than one is possible, especially if different version numbers of the same server are installed on a single host.

The default server root in Unix is `usr/netscape/server4/` and in Windows NT is `C:\Netscape\Server4`.

Product Name. The complete product name.

Vendor. The name of the vendor.

Version. The version number.

Build Number. The number that identifies the build that was used for this installation.

Security Level. The server's security level—whether the server is meant for use in the United States and Canada (domestic) or any other part of the world (export). (See "Configuring the Server's Security Preferences" on page 268.)

Server Status. The server's status—whether it is `started`, `stopped`, or `unknown`; normally, `unknown` indicates that the server hasn't been configured properly.

Changing the Name of an Instance

Following installation, the name of a CMS instance is in the form:

`CMS` (`cert-<instance_id>`)

`<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.

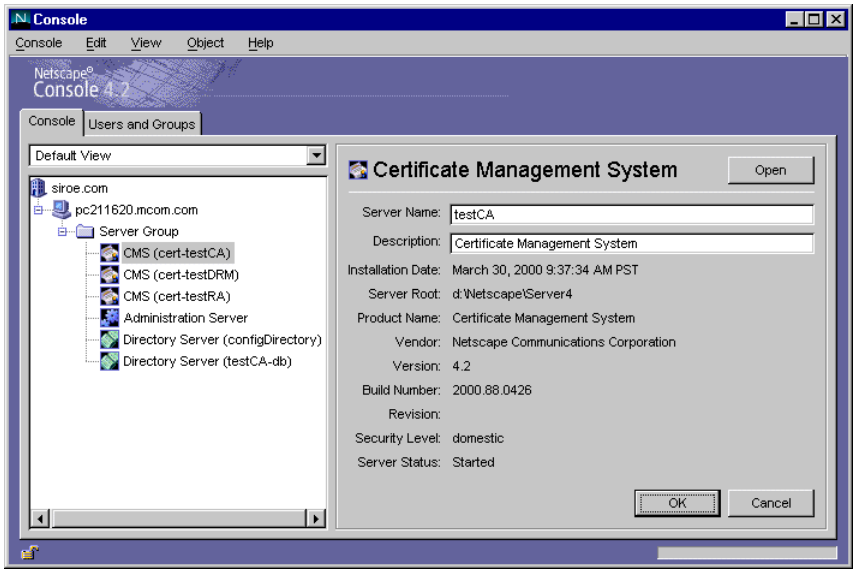
For example, if you installed an instance of Certificate Management System with an ID of `testCA`, the instance name will be `cert-testCA`.

You can change the instance name to a more descriptive one later. If you change the instance name, Certificate Management System uses the new name as a descriptive nickname for the instance. It shows the new name in Netscape Console only; it does not change the original instance ID in the configuration.

To change the name of a particular CMS instance:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. In the Console tab, select the CMS instance you want to rename.
3. Click Edit.

Details about the selected CMS instance appear in the right pane.



4. Specify the appropriate information:

   Server Name. Type a descriptive name for the server.

   Description. Type any additional description for the server. For example, you may want to type information that will help you identify this instance of Certificate Management System.

5. Click OK.

   You are returned to the previous screen. The new name appears in the right pane.

Removing an Instance From a System

If you are sure you won't need a particular CMS instance anymore, you can use Netscape Console to remove the server instance from your machine. Removing a CMS instance is not the same as uninstalling Certificate Management System; when you uninstall Certificate Management System, its program files are deleted from the host machine. (For instructions, see "Uninstalling Certificate Management System" on page 123.)

To remove a CMS instance from your machine:

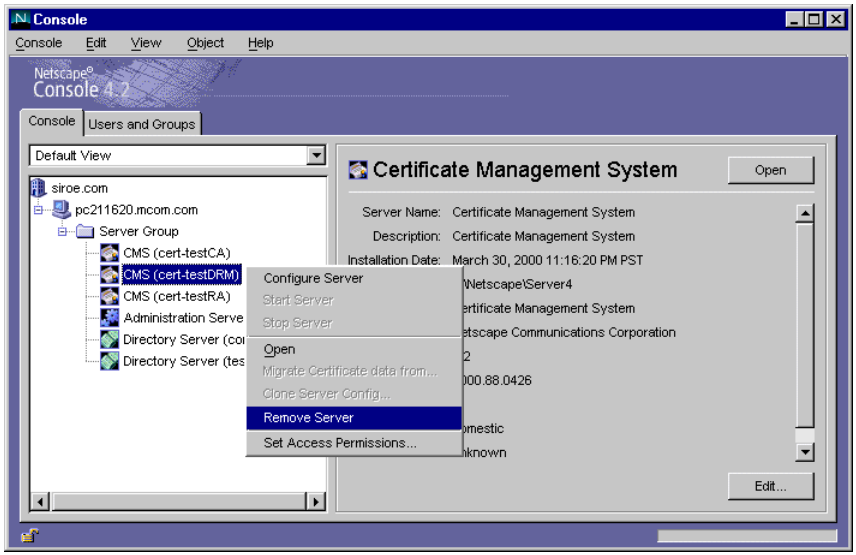
1. Log in to Netscape Console (see “Logging In to Netscape Console” on page 69).

2. In the Console tab, select the CMS instance you want to remove.

3. From the Object menu, choose Stop; you can also right-click to choose this option from the pop-up menu (see the figure below).

4. When the server has stopped, from the Object menu, choose Remove Server.

   As shown in the figure below, you can also right-click to choose this option from the pop-up menu.



5. When prompted, confirm that you want to remove the server instance.

   The selected CMS instance is removed. The corresponding internal database is not removed. If you want to remove it, select the instance, and repeat steps 3 through 5.

   The Directory Server (configuration directory) and Administration Server binaries are also not removed; you require these to administer the remaining servers installed in the same server group.

Uninstalling Certificate Management System

To remove files pertaining to Certificate Management System from a host system, run the uninstallation program. Uninstalling Certificate Management System removes all the corresponding CMS instances from the navigation tree of Netscape Console. To remove a specific CMS instance, follow the instructions provided in "Removing an Instance From a System" on page 121.

You can uninstall Certificate Management System in two ways:

- From the command line (locally only)
- On a Windows NT system, by using the Windows NT Add/Remove Programs Utility

Uninstalling from the Command Line

To uninstall Certificate Management System from the command line:

1. Open a terminal window to your server.
2. In a Unix system, log in either as `root` or using the server's user account (if that is how you started the server).
3. At the command-line prompt, enter the following line:
   - On Windows NT, enter `<server_root>\uninst`.
   - On Unix, enter `<server_root>/uninstall`.

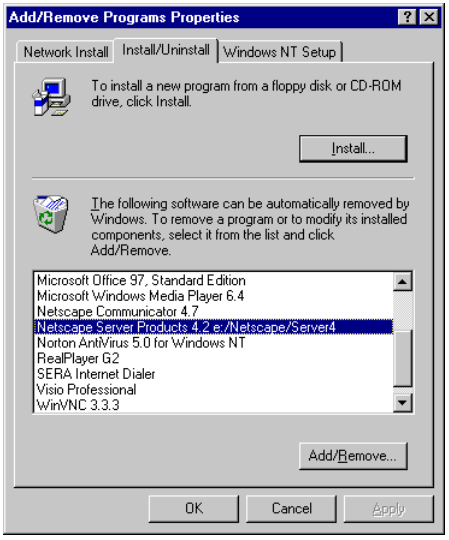
   `<server_root>` is the directory where the CMS binaries are kept. You first specified this directory during installation.

   The uninstallation program starts.

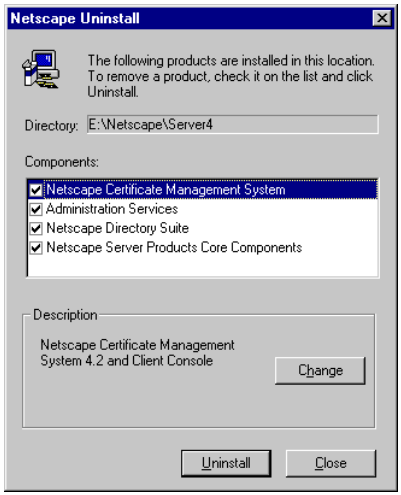
Uninstalling by Using the Windows NT Add/Remove Programs Utility

To remove Certificate Management System by using the Windows NT Add/Remove Programs utility:

1. From the Start menu, choose Settings, then Control Panel.
2. In the Control Panel, choose Add/Remove Programs.
3. In the Add/Remove Programs Properties window, choose `Netscape Server Products 4.2 <server_root>`, and click Add/Remove.



4. In the Netscape Server Uninstall window, make sure all the components are selected, and click Uninstall.



The uninstallation program starts.

Chapter

5

Starting and Stopping CMS Instances

This chapter describes how to start, stop, and restart Netscape Certificate Management System (CMS) and how to check its current status. The chapter also explains the CMS watchdog process, a native bootstrapping program that enables Certificate Management System to start up with a single password instead of multiple ones.

The chapter has the following sections:

Note You can use the CMS window only when the appropriate Administration Server is running. Be sure to start Administration Server at the port you specified during CMS installation. To minimize security risks, shut down Administration

Server when you have finished using Netscape Console. For instructions on starting and shutting down Administration Server, see "Netscape Administration Server" on page 66.

Starting Certificate Management System

Once Certificate Management System is installed, it runs constantly, listening for and accepting requests. You can start Certificate Management System in several ways:

- From Netscape Console (locally and remotely)
- From the command line (locally only)
- On a Windows NT system, from the Windows NT Services panel

Required Start-up Information

When you start Certificate Management System, you are prompted to enter the *single sign-on* password you specified during installation. This password enables the CMS watchdog (see "CMS Watchdog Process" on page 143) to retrieve all the passwords required by the server to start. These include the following:

- Passwords for the internal or external (if any are currently installed) tokens; these tokens contain certificates and corresponding public and private key pairs for the server.
- The bind password used by Certificate Management System to access and update the internal database.
- The bind password used by Certificate Management System to access and remove PINs from the authentication directory, if you've configured Certificate Management System to remove PINs from the authentication directory (see the description for the `ldap.ldapauthbindDN` and `ldap.ldapauth.bindPWPrompt` parameters on Table 10.3 on page 334).

- The bind password used by Certificate Management System to access and create/modify user entries in the directory used for portal registration, if you've configured Certificate Management System for portal enrollment (see the description for the `ldap.ldapauthbindDN` and `ldap.ldapauth.bindPWPrompt` parameters on Table 10.5 on page 354).

- The bind password used by Certificate Management System to access and update the LDAP directory; this is required only if you have configured Certificate Management System for publishing certificates and CRLs to an LDAP-compliant directory.

You first specified these passwords when you installed Certificate Management System. Keep in mind that the passwords you provide for the tokens unlock a combination of the following private keys:

- If you have installed a Certificate Manager in the currently selected CMS instance, the token password unlocks the private keys for the Certificate Manager's *CA signing* and *SSL server* certificates.

- If you have installed a Registration Manager in the currently selected CMS instance, the token password unlocks the private keys for the Registration Manager's *signing* and *SSL server* certificates.

- If you have installed a Data Recovery Manager in the currently selected CMS instance, the token password unlocks the private keys for the Data Recovery Manager's *storage* keys and *transport* and *SSL server* certificates.

For more information about the CMS keys and certificates, see "Keys and Certificates" on page 225.

Note that during CMS installation, the watchdog stores all the passwords, required by the server for starting up, in a password cache. The cache is maintained in a file encrypted using the single sign-on password you specify during installation. When you change any of the required passwords or provide new passwords, you must start the server from the command-line (see "Starting From the Command Line" on page 135) so that the watchdog can prompt you for the new passwords in order to update the cache.

The single sign-on password eliminates the need for you to enter the various password when starting up Certificate Management System. As a security measure, you should consider changing the single sign-on password periodically. For instructions, see "Changing the Single Sign-On Password" on page 148.

Also note that all passwords used in Certificate Management System are checked by a built-in password-quality checker; for details, see "Password-Quality Checker" on page 151.

Configuring the Server to Start Without the Single Sign-On Password

If you prefer to start up Certificate Management System by entering all the required passwords, instead of just the single sign-on password, you can do so by either deleting or renaming the password cache file, `pwcache.p12` (notice the `.p12` extension).

Here's how you can do it:

1. Go to this directory: `<server_root>/cert-<instance_id>/config`
2. Locate the `pwcache.p12` file.
3. Either rename or delete the file.
4. Start the server from the command line; see "Starting From the Command Line" on page 135.

   You are prompted for all the required passwords.

Later, if you want to revert back to starting the server using the single sign-on password:

1. Create a password cache by following the instructions in "Creating a New Password Cache" on page 150.
2. Create entries for all the required passwords by following the instructions in "Adding a New Entry to the Password Cache" on page 149.
3. Copy the file to the `<server_root>/cert-<instance_id>/config` directory.
4. Start the server from the command line; see "Starting From the Command Line" on page 135.

   You are prompted for the single sign-on password.

Configuring the Server to Read the Single Sign-on Password From a File

Every time you start Certificate Management System, you are required to enter either the single sign-on password or all the passwords required by the server to startup (see "Required Start-up Information" on page 128 and "Configuring the Server to Start Without the Single Sign-On Password" on page 130). If it is inconvenient for you to start the server this way, you can store the single sign-on password in a file and configure the server to start by reading the password from that file.

This configuration eliminates the need for you to enter the single sign-on password every time you start the server. Also, if you have remote access to the rebooted system, you can restart the server without sending the single-sign-on password over the net.

Caution The instructions that follow explain how to configure Certificate Management System to start by reading the single sign-on password from a file. Note that the password is stored in a *plain text* file and you must use your operating system's security feature to secure this file. Failing to do so poses a security risk, as anyone who has access to the host system will be able to get hold of the single sign-on password.

To configure the server to start by reading the single sign-on password from a file:

1. Create a file named `pwfile`.
2. Put the single sign-on password in the file.
3. Copy the file to the `<server_root>/cert-<instance_id>/config` directory.
4. Edit the `start-cert` script.

   To edit the `start-cert` script in Unix, follow these steps:

   1. Open a command-line window.
   2. Go to the CMS-instance directory. For example, `/usr/netscape/server4/cert-testCA`.

3. Enter the following line at the prompt:

   ```
   cat start-cert
   ```

   You should see something similar to this:

   ```
   #!/bin/sh
   /usr/netscape/server4/bin/cert/admin/bin/start -i testCA
   -r /usr/netscape/server4 -e -classpath
   /usr/netscape/server4/bin/cert/classes:/usr/netscape/
   server4/bin/cert/jars/jss.jar:/usr/netscape/server4/bin/
   cert/jars/certsrv.jar:usr/netscape/server4/java/
   ldapjdk.jar:/usr/netscape/server4/bin/cert/jre/lib/
   rt.jar:/usr/netscape/server4/bin/cert/jre/lib/i18n.jar:/
   usr/netscape/server4/bin/cert/jars/jssjdk12.jar
   ```

4. Edit the script to include the file path to the `pwfile` file. Be sure to include the file path as shown in the example (shown in bold).

   ```
   #!/bin/sh
   /usr/netscape/server4/bin/cert/admin/bin/start -i testCA
   -f config/pwfile -r /usr/netscape/server4 -e -classpath
   /usr/netscape/server4/cert-testCA/classes/:/usr/netscape/
   server4/bin/cert/classes/:/usr/netscape/server4/bin/cert/
   jars/jss.jar:/usr/netscape/server4/bin/cert/jars/
   certsrv.jar:/usr/netscape/server4/java/ldapjdk.jar:/usr/
   netscape/server4/bin/cert/jre/lib/rt.jar:/usr/netscape/
   server4/bin/cert/jre/lib/i18n.jar:/usr/netscape/server4/
   bin/cert/jars/jssjdk12.jar
   ```

To edit the `start-cert.bat` script in Windows NT, follow these steps:

1. Open a command-line window.

2. Go to the CMS instance directory. For example, `C:\netscape\server4\cert-testCA`.

3. Enter the following line at the prompt:

   ```
   type start-cert.bat
   ```

   You should see something similar to this:

   ```
   net start cert-testCA /cC:\Netscape\Server4\cert-
   testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\
   Netscape\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server
   ```

```
4\bin\cert\jars\certsrv.jar;C:\Netscape\Server4\java\ldap
jdk.jar;C:\Netscape\Server4\bin\cert\jre\lib\rt.jar;C:\Ne
tscape\Server4\bin\cert\jre\lib\i18n.jar;C:\Netscape\Serv
er4\bin\cert\jars\jssjdk12.jar;C:\Netscape\Server4\java\s
wingall.jar
```

4. Edit the script to include the file path to the `pwfile` file. Be sure to include the file path as shown in the example (shown in bold).

   ```
   net start cert-testCA /fC:\Netscape\Server4\cert-
   testCA\config\pwfile /cC:\Netscape\Server4\cert-
   testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\
   Netscape\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server
   4\bin\cert\jars\certsrv.jar;C:\Netscape\Server4\java\ldap
   jdk.jar;C:\Netscape\Server4\bin\cert\jre\lib\rt.jar;C:\Ne
   tscape\Server4\bin\cert\jre\lib\i18n.jar;C:\Netscape\Serv
   er4\bin\cert\jars\jssjdk12.jar;C:\Netscape\Server4\java\s
   wingall.jar
   ```

5. Save your changes.

5. Use your operating system's security feature to restrict access to the password file.

6. Restart the server from the command line; see "Starting From the Command Line" on page 135.

   It should start without prompting for the single sign-on password.

Starting From Netscape Console

You can use Netscape Console to start an instance of Certificate Management System running on a local or remote host.

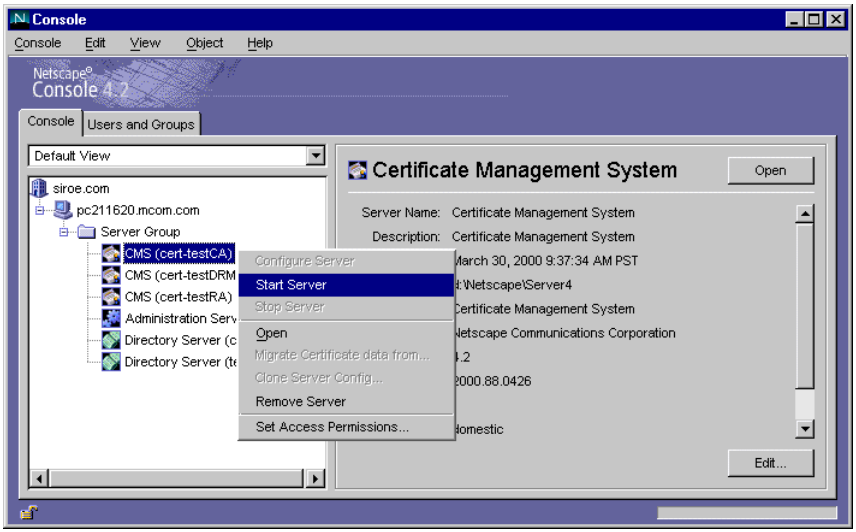
To start Certificate Management System from Netscape Console:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).

2. In the Console tab, select the Server Group that contains the CMS instance you want to start.

3. In the navigation tree, locate the CMS instance you want to start.

4. Select the instance, right-click, and select the Start Server option from the pop-up menu.



When you start Certificate Management System, you are prompted to supply the single sign-on password for the server.

5. Type the single sign-on password you specified during installation and click OK.

   Certificate Management System won't start until you provide this password. For more information, see "Required Start-up Information" on page 128.

Starting From the Command Line

To start Certificate Management System from the command line:

1. Open a terminal window to your server.
2. In a Unix system, log in as `root` if the server runs on ports less than 1024; otherwise, log in either as `root` or with the server's user account.
3. At the command-line prompt, enter the following line:

   ```
   <server_root>/cert-<instance_id>/start-cert[.bat]
   ```

   `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

   `<server_root>` is the directory where the CMS binaries are kept. You first specified this directory during installation.

   `<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.
4. When prompted, enter the single sign-on password.

   Certificate Management System won't start until you provide this password. For more information, see "Required Start-up Information" on page 128.

Note If Certificate Management System is already running, the start-up command fails. Stop the server first using the `stop-cert` command, then use the `start-cert` command.

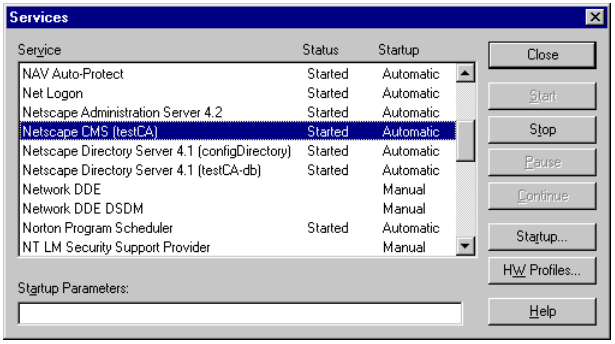
Starting From the Windows NT Services Panel

If you have installed Certificate Management System on a Windows NT system, you can start the server (as a service) from the Windows NT Services panel (see Figure 5.1). The CMS service has the following name:

```
Netscape Certificate Management System (cert-<instance_id>)
```

`<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.

Figure 5.1 CMS service in the Windows NT Services panel



To start Certificate Management System from the Windows NT Services panel:

1. Click the Start button on your desktop.
2. Select Control Panel from Settings.
3. In the Control Panel window that appears, click Services.
4. Select the CMS instance and click Start.

   You are prompted to supply the single sign-on password for the server.
5. Enter the single sign-on password you specified during installation and click OK.

   Certificate Management System won't start until you provide this password. For more information, see "Required Start-up Information" on page 128.

Stopping Certificate Management System

You can stop Certificate Management System in several ways:

- From Netscape Console (locally and remotely)
- From the command line (locally only)
- On a Windows NT system, from the Windows NT Services panel

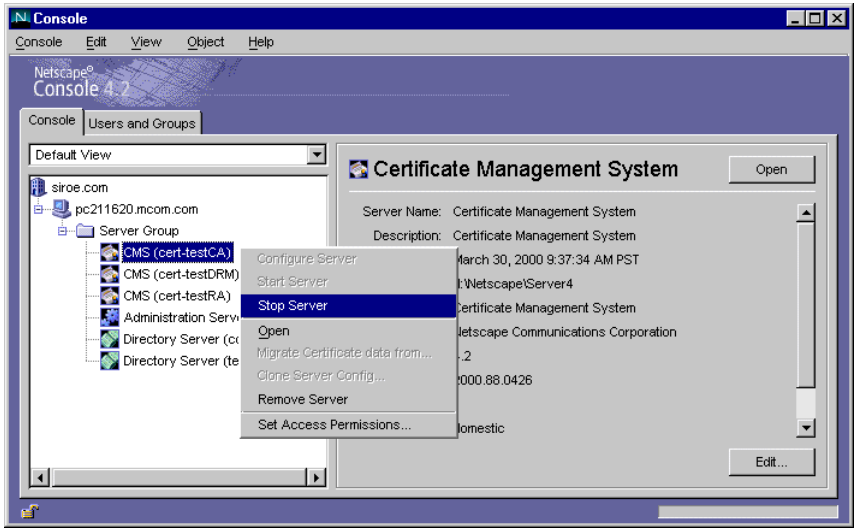
Stopping Certificate Management System shuts down all the subsystems completely, interrupting service until the server is started again. If your machine crashes or is taken offline, the server stops, and any requests it was servicing are lost. You need to start the server again to restore service.

Stopping From Netscape Console

You can use Netscape Console to stop an instance of Certificate Management System running on a local or remote host.

To stop Certificate Management System from Netscape Console:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. In the Console tab, select the Server Group that contains the CMS instance you want to stop.
3. In the navigation tree, select the CMS instance you want to stop, right-click, and select the Stop Server option from the pop-up menu.



The server is stopped.

Stopping From the Command Line

You can stop a CMS instance running on a local host by entering the appropriate command at the command prompt.

To stop a Certificate Management System from the command line:

1. Open a terminal window to your server.
2. In a Unix system, log in either as `root` or using the server's user account (if that is how you started the server).
3. At the command-line prompt, enter the following line:

   `<server_root>/cert-<instance_id>/stop-cert[.bat]`

   `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

   `<server_root>` is the directory where the CMS binaries are kept. You first specified this directory during installation.

   `<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.

   The server is stopped.

Stopping From the Windows NT Services Panel

You can stop a CMS instance running on a local host by stopping the corresponding service; it is identified by the following in the Windows NT Services panel (see Figure 5.1 on page 136):

`Netscape Certificate Management System (cert-<instance_id>)`

`<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.

To stop Certificate Management System from the Windows NT Services panel:

1. Click the Start button on your desktop.
2. Select Control Panel from Settings.
3. In the Control Panel window that appears, click Services.
4. Select the CMS instance and click Stop.
5. When prompted, click Yes.

   The server is stopped.

Restarting Certificate Management System

Whenever you change the CMS configuration, you must save your changes (by clicking the Save button) for the changes to take effect. Some configuration changes also require that you *restart* the server after you save the changes. If restarting is required, the server prompts you accordingly.

You can restart the server in two ways:

- From the CMS window (locally and remotely)
- From the command line (locally only)

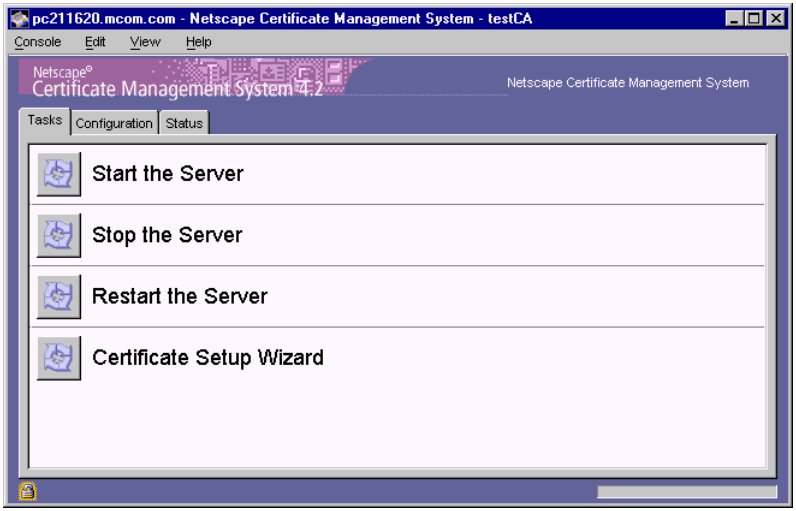
Restarting From the CMS Window

You can use the CMS window to restart an instance of Certificate Management System on a local or remote host.

To restart Certificate Management System from the CMS window:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. In the Tasks tab, click Restart the Server.



When you restart Certificate Management System, you are prompted to supply the single sign-on password for the server.

3. Type the single sign-on password you specified during installation and click OK.

   Certificate Management System won’t restart until you provide this password. For more information, see “Required Start-up Information” on page 128.

Restarting From the Command Line

To restart Certificate Management System from the command line:

1. Open a terminal window to your server.
2. In a Unix system, log in either as `root` or using the server's user account (if that is how you started the server).
3. At the command-line prompt, enter the following line:

   `<server_root>/cert-<instance_id>/restart-cert[.bat]`

   `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

   `<server_root>` is the directory where the CMS binaries are kept. You first specified this directory during installation.

   `<instance_id>` is the ID for this instance of Certificate Management System. You first specified this when you installed this server.
4. When prompted, enter the single sign-on password.

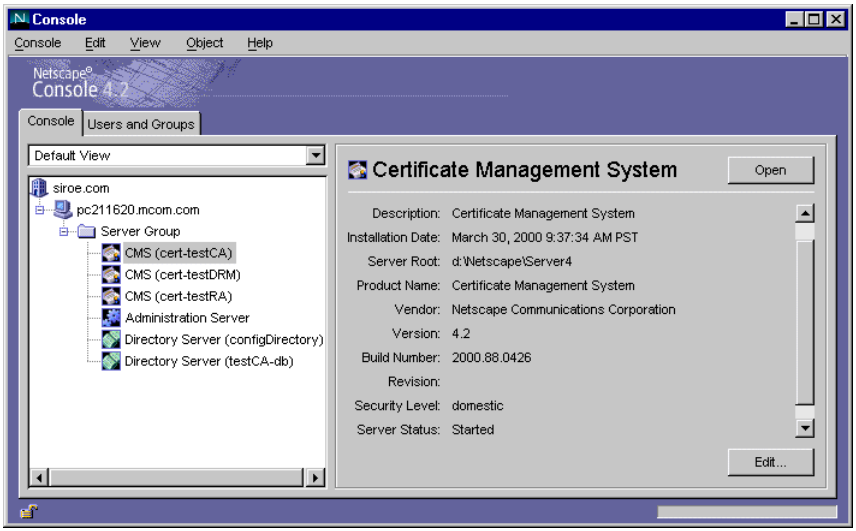
   Certificate Management System won't restart until you provide this password. For more information, see "Required Start-up Information" on page 128.

Checking System Status

You can use Netscape Console to find out whether a particular instance of Certificate Management System is running.

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. In the Console tab, select the instance that corresponds to the CMS instance you want to check.

In the right pane, check the Server Status field. If the selected instance of Certificate Management System is running, the status will be *Started*. Otherwise it will be *Stopped* or *Unknown*.



Attending to an Unresponsive Server

If an error causes Certificate Management System to become unresponsive, and all attempts to stop it from Netscape Console fail, it may be necessary to kill the server processes manually. The processes that should be killed are identified as follows: `cms_server`, `cms_watchdog`, or `cms_daemon`.

On a Windows NT system, the server processes will have `.exe` file extension and will be listed in the Windows NT Task Manager. However, because they are system processes, you cannot terminate them from the Task Manager. Instead, you should terminate them using the `killproc` command-line tool. This tool is located with the rest of the command-line tools provided with Certificate Management System:

```
<server_root>/bin/cert/tools
```

In order to kill system processes, the user that runs `killproc` must have the *Debug Programs* permission. By default, this permission is given only to the Administrators group, although this can be changed in the Windows NT User Manager. Assuming it is not changed, `killproc` must be run by a member of the Administrators group (such as the user `Administrator`).

The `killproc` command takes one argument, the process ID of the process to be killed: `killproc <process_id>`

You can obtain the process ID from the Windows NT Task Manager. For example, to kill the `jre` process whose process ID is 255, you should type:

```
c:\> killproc 255
Killed process 255.
c:\>
```

Note The `killproc` tool should only be used as a last resort. Because it forces the process to terminate abruptly, the process is not able to cleanup or to save its internal state before exiting.

CMS Watchdog Process

The CMS watchdog is a native bootstrapping program that provides specific native functions. It works with Certificate Management System to enable it to start up using a single password—instead of multiple passwords—called the *single sign-on* password. In addition, it manages the start-up, stop, and restart states of Certificate Management System.

The watchdog process (identified as `cms_watchdog`) implements the following operations:

- Starts Certificate Management System (and the Virtual Java Machine, or Java VM).

  The watchdog allows you to start Certificate Management System by using a single password instead of the multiple passwords that would otherwise be required. For details on these passwords, see "Required Start-up Information" on page 128.

  During CMS installation the watchdog stores all the passwords required by the server for starting up in a password cache, which is explained in "Password Cache" on page 144.

- Stops Certificate Management System.
- Restarts Certificate Management System (after configuration changes).
- Detects Certificate Management System crashes and restarts the server.

  The watchdog monitors Certificate Management System and the Java VM, restarting the server in the case of a failure.
- In the Unix version of Certificate Management System, the watchdog records the server process ID (`pid`) and sets the user ID (`uid`) of the process.

Password Cache

During CMS installation, the installation program creates a password cache which the CMS watchdog uses to store all the passwords required by the server during start up (see "Required Start-up Information" on page 128). For example, when you specify the cryptographic token password and the bind password for the internal directory during installation, the watchdog adds these passwords into the password cache; similarly, when you configure the server for LDAP publishing from Netscape Console, the watchdog adds the corresponding password to the cache.

The password cache is maintained in a triple-DES encrypted file named `pwcache.p12`, which is located here:

```
<server_root>/cert-<instance_id>/config
```

The file is protected using the single sign-on password you specify during installation. In the cache, passwords are stored along with a name, a string describing the usage of the password, which is used by Certificate Management System to index into the cache. For example, the contents of the password cache could look like this:

```
----- Password Cache -----
Internal LDAP Database : myIdbPwd
Internal Key Storage Token : myTokenPwd
Authentication : myPinAuthPwd
LDAP Publishing : myLdapPubPwd
```

Note that in the above example

- The string `Internal LDAP Database` is the default value assigned to the `internaldb.ldapauth.bindPWPrompt` parameter in the CMS configuration file; it provides a descriptive usage for the password Certificate Management System uses to bind to the internal database.
- The string `Internal Key Storage Token` is hardcoded and it refers to the Netscape Software Cryptographic Service provider; you cannot change it. You can only change the corresponding password.

Other entries may appear in the password cache. For example, if you set up PIN-based authentication with the remove PIN option, you will see an entry for the password Certificate Management System uses to bind to the authentication directory to remove a PIN after a user successfully authenticates; for details, see Table 10.3 on page 334. Similarly, if you enable LDAP publishing with *basic authentication*, you will also see an entry for the password Certificate Management System will use to bind to the publishing directory; for details, see "Step 5. Identify the Publishing Directory" on page 830.

Except for the string `Internal LDAP Database`, you can change any of the above prompts by modifying the corresponding value in the configuration file and then replacing (delete the old item and add the new item) the current entry in the password cache with the new prompt and the password using the `PasswordCache` utility explained in "Password Cache Utility" on page 146.

When various modules in the server, such as authentication and LDAP publishing, initialize, they query the password cache for the password. The password cache returns the password if it has it, or else it prompts the user for one. Note that this prompting happens only at server startup time, which means whenever you change any of the required passwords or provide new passwords, you must restart the server from the command-line (see "Starting From the Command Line" on page 135) so that the watchdog can prompt you for the new passwords in order to update the cache.

Password Cache Utility

Certificate Management System comes with a command-line utility named `PasswordCache` for manipulating the contents of the password cache. You will be required to manipulate the password cache for various reasons. For example, assume you've enabled LDAP publishing and have configured

Certificate Management System to bind to the directory with Directory Manager's DN and password. If the directory administrator changes the Directory Manager's password, Certificate Management System will fail to bind to the directory during startup. You can resolve this problem by modifying the corresponding bind password in the cache using the `PasswordCache` utility.

Locating the PasswordCache Utility

The `PasswordCache` utility is located with the rest of the command-line tools here: `<server_root>/bin/cert/tools`

Note You must run the `PasswordCache` utility from the `<server_root>/cert-<instance_id>` directory.

Syntax

You can run the utility by executing the following command from the `<server_root>/cert-<instance_id>` directory:

```
PasswordCache <sso_password> <command>
```

where `<sso_password>` specifies the current single sign-on password and `<command>` can be any of the following:

- `list`
- `add <password_name> <password>`
- `change <password_name> <password>`
- `delete <password_name>`
- `changesso <new_sso_password>`
- `create`

`<password_name>` specifies the string (describing the password usage) you want to add to, or modify or delete from the cache; it is equivalent to the value assigned to the `bindPWPrompt` or `tokenname` parameter in the CMS configuration file.

`<password>` specifies the new password.

`<new_sso_password>` specifies the new single sign-on password.

Managing the Password Cache

You can use the `PasswordCache` utility for the following:

- Changing the single sign-on password
- Listing or viewing the contents of the password cache
- Adding a new entry to the cache
- Changing the password associated with an entry
- Deleting an entry in the cache

The sections that follow explain how you can accomplish the above mentioned tasks.

Note The server queries the password cache only during start up, and hence recongnizes the changes you've made to the cache only if you restart the server from the command line. If you left any of the passwords blank, the server will prompt you to enter that during startup and from then on stores it in the password cache.

Changing the Single Sign-On Password

To change the single sign-on password:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with the single sign-on password and `<new_sso_password>` with the new single sign-on password.

   ```
   PasswordCache <sso_password> changesso <new_sso_password>
   ```

   For example, if your old password is `mySsoPwd` and new password is `myNewSsoPwd`, the command would look like this:

   ```
   PasswordCache mySsoPwd changesso myNewSsoPwd
   ```

Listing the Contents of the Password Cache

To list or view the contents of the password cache:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with the single sign-on password:

   ```
   PasswordCache <sso_password> list
   ```

   For example, if your single sign-on password is `mySsoPwd`, the command would look like this:

   ```
   PasswordCache mySsoPwd list
   ```

   In response, you should see something similar to this:

   ```
   ----- Password Cache -----
   Internal LDAP Database : myIdbPwd
   Internal Key Storage Token : myTokenPwd
   LDAP Publishing: myLdapPubPwd
   ```

Adding a New Entry to the Password Cache

To add a new entry to the cache:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with the single sign-on password, `<password_name>` with a string describing the password usage, and `<password>` with the actual password:

   ```
   PasswordCache <sso_password> add <password_name> <password>
   ```

   For example, if your single sign-on password is `mySsoPwd`, the string describing the password usage is `Bind Password for LDAP Publishing Directory`, and password is `myLdapPubPwd`, the command would look like this:

   ```
   PasswordCache mySsoPwd add "Bind Password for LDAP Publishing Directory" myLdapPubPwd
   ```

If the password name string includes spaces, be sure to enclose the string in double quotes as indicated in the above example.

Changing the Password of an Entry in the Password Cache

To change the password associated with an entry in the password cache:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with the single sign-on password, `<password_name>` with the string that describes the password usage, and `<password>` with the new password:

   ```
   PasswordCache <sso_password> change <password_name>
   <password>
   ```

   For example, if your single sign-on password is `mySsoPwd`, the string describing the password usage is `Bind Password for LDAP Publishing Directory`, and the new password is `myNewLdapPubPwd`, the command would look like this:

   ```
   PasswordCache mySsoPwd change "Bind Password for LDAP
   Publishing Directory" myNewLdapPubPwd
   ```

   If the password name string includes spaces, be sure to enclose the string in double quotes as indicated in the above example.

Deleting an Entry From the Password Cache

To delete an entry from the cache:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with the single sign-on password and `<password_name>` with the string that describes the password usage:

   ```
   PasswordCache <sso_password> delete <password_name>
   ```

For example, if your single sign-on password is `mySsoPwd` and the string describing the password usage is `Bind Password for LDAP Publishing Directory`, the command would look like this:

```
PasswordCache mySsoPwd delete "Bind Password for LDAP Publishing Directory"
```

If the password name string includes spaces, be sure to enclose the string in double quotes as indicated in the above example.

Creating a New Password Cache

If you have changed CMS startup so that the server prompts for all the required passwords, instead of just the single sign-on password, and want to revert back to starting the server with a single sign-on password, you must create a new password cache. Before creating a new password cache, decide on the single sign-on password to protect the cache.

To create a new, empty password cache:

1. Open a command window.
2. Go to this directory: `<server_root>/cert-<instance_id>`
3. At the prompt, enter the command below, substituting `<sso_password>` with a password to protect the cache:

   ```
   PasswordCache <sso_password> create
   ```

   For example, if the password you want to use to protect the single sign-on cache is `mySsoPwd`, the command would look like this:

   ```
   PasswordCache mySsoPwd create
   ```

Password-Quality Checker

Certificate Management System comes with a plugin, called *password-quality checker*, to monitor the quality of passwords set within the CMS system. All passwords used in Certificate Management System are checked by the password-quality checker, which by default checks that the length of a password is at least 8 characters long; there are no checks regarding which characters are valid or invalid. If you use a password that doesn't meet the quality rules, you will get an error message indicating that the password didn't meet the password-quality rules.

Note that Certificate Management System enforces password quality on only those passwords that it strictly creates and manages. Passwords you enter for LDAP directory access are not subjected to quality checks. The reason for this is, the password quality is handled by the system that creates and manages the password. In an LDAP directory access, the remote directory that you authenticate to enforces the password quality of the password you use because it is created and managed by the directory.

To enable you to customize password quality, the plugin for the password-quality checker is included in the CMS samples package; for example, you can change the default rule to ensure that all CMS passwords are constructed with certain types of characters such as numbers, symbols, capital letters, and so on. The samples package is located here:

```
<server_root>/cms_sdk/samples
```

3

System-Level Configuration

Chapter

6

Configuring Ports, Database, and SMTP Settings

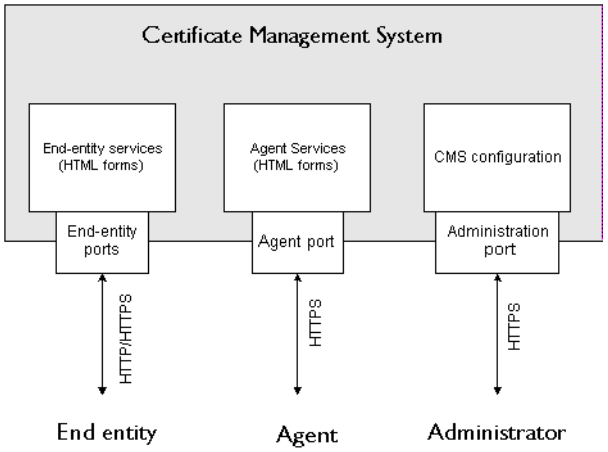
Subsystems installed in an instance of Netscape Certificate Management System (CMS) share certain configuration information. They use the same administration, agent, and end-entity ports; internal database for data storage; mail server for automated notifications; internal token and trust database for PKI operations; SSL ciphers during SSL negotiation; privileged users; and log files to log messages to. This chapter explains how to configure these ports, the internal database, and the mail server settings for a CMS instance.

The chapter has the following sections:

CMS Ports

Certificate Management System listens to different ports for requests from different users. As illustrated in Figure 6.1, it listens to the administration port, the agent port, and end-entity ports.

Figure 6.1 CMS ports for administration, agent, and end-entity operations



When choosing ports for Certificate Management System, be sure to choose ports that are unique on the host system—that is, no other application can be using, or attempting to use, the port numbers you assign to Certificate Management System. To verify that a port is available for use, check the appropriate file for your operating system; port numbers for network-accessible services are usually maintained in a file named `services`. (On Unix, if you are not running as `root` or `superuser` when you install or start the server, you will have to use a port number higher than 1024.)

Remote Administration Port

The administration port is an SSL (encrypted) port at which Certificate Management System listens to requests from its administration interface; administrators make these requests from the CMS window. When you install Certificate Management System, it assigns a random number (greater than 1024) as the administration port number. You can change this port number at any time, to any number between 1 and 65535. For security reasons you should consider changing the administration port number periodically.

Agent Port

The agent port is an SSL (encrypted) port at which Certificate Management System listens to requests from agents; agents make these requests from the appropriate Agent Services interface.

- The Certificate Manager and Registration Manager agents use the agent port to process certificate issuance and management requests from end entities and to perform certain other privileged operations over HTTPS.
- Data Recovery Manager agents use the agent port for recovering end users' encryption private keys over HTTPS.

Agent functions always require SSL client authentication. For a list of supported agent operations, see "Agent Services" on page 899.

When you install Certificate Management System, it assigns a random number (greater than 1024) as the agent port number and prompts you to change it, if necessary; the port number can be any number between 1 and 65535. The number you choose for the agent port affects your agent users—all agents access Certificate Management System by specifying the name of the server (the CMS instance) and the agent port number in the URL. For example, if you choose port number 4430, the URL would look like this:

`https://<host_name>:4430/<subsystem>`

`<host_name>` is in the form `<machine_name>.<your_domain>.<domain>`

`<subsystem>` is a prefix identifying the subsystem that hosts the agent interface:

- `ca` for the Certificate Manager
- `ra` for the Registration Manager
- `kra` for the Data Recovery Manager

For example, the URL to a Certificate Manager agent interface would look like this: `https://testCA.netscape.com:5600/ca`

If you change the agent port number, be sure to inform your agent users.

End-Entity Ports

For requests from end entities, Certificate Management System can listen to two ports, an SSL (encrypted) port and a non-SSL port. End entities make these requests from the end entity services interface; see "End-Entity Services" on page 895.

Certificate Management System provides the following services through the HTTP and HTTPS ports:

- The HTTP port can be used to service end-entity-initiated PKI requests, such as enrollment, renewal, and revocation; enrollment requests can include requests from Cisco routers (using the CEP protocol). You have the choice of keeping this port enabled or disabled.
- The HTTPS port can be used to provide the following services for enforcing data privacy and client authentication:
  - — End-entity-initiated PKI requests, such as enrollment, renewal, and revocation
  - — General certificate retrieval requests, such as retrieving a single certificate identified by a serial number, listing certificates based on certain criteria (for example, an LDAP search filter defined over standard attributes), and getting a CA's certificate chain

  Similar to the HTTP port, you can enable or disable the HTTP port. For example, if you don't want end-entity interaction with a Certificate Manager, you can disable the HTTPS port. For details, see "Step 6. Enable End-Entity Interaction" on page 405.

Configuring Port Numbers

Configuring port numbers for a CMS instance involves two steps:

- Step 1. Specify the Port Number
- Step 2: Specify IP Addresses

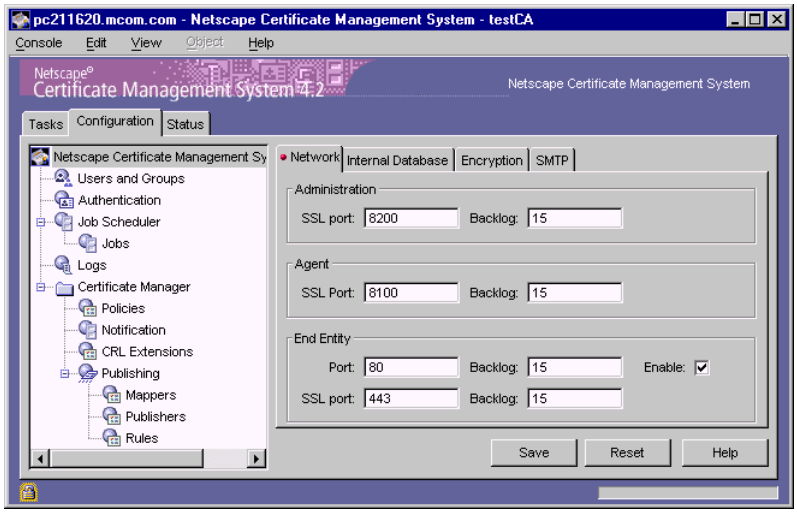
Step 1. Specify the Port Number

To change the administration, agent, or end-entity port numbers used by a CMS instance:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. Select the Configuration tab.

   The Network tab appears.



3. To change the administration port number, enter the port number in the Administration section:

   SSL port. Type a TCP/IP port number. Certificate Management System uses this port for SSL-enabled communications with the CMS window—that is, HTTPS requests from administrators. Make sure the port number you specify is unique on the host system.

   Backlog. Type the number of connections that can be waiting to be serviced at the administration port. The default number is 15. The number you enter in this field is passed to the operating system’s `listen()` call.

4. To change the agent port number, enter the port number in the Agent section:

   SSL port. Type a TCP/IP port number. Certificate Management System uses this port for SSL-enabled communications with the Agent Services interface—that is, HTTPS requests from agents. Make sure the port number you specify is unique on the host system.

   Backlog. Type the number of connections that can be waiting to be serviced at the agent port. The default number is 15. The number you enter in this field is passed to the operating system's `listen()` call.

5. To change the end-entity port numbers, enter the port numbers in the End Entity section.

   Certificate Management System is capable of simultaneous SSL and non-SSL communications at the end-entity port. This means that you do not have to choose between SSL and non-SSL communications; you can use both at the same time. But if you prefer, you can disable the non-SSL port by unchecking the "Enable" option.

   Port. Type a TCP/IP port number that is unique on the host system. Certificate Management System uses this port for non-SSL communications with the end entity services interface.

   This port is provided to allow enrollments of end entities that do not support SSL; for example, HTTP requests from end entities such as routers. You can use the Enable check box to turn this port on or off. Keep in mind that if this port is enabled, end entities will be able to enroll over HTTP too, which means their certificate requests could be intercepted and replayed to the server.

   Backlog. Type the number of connections that can be waiting to be serviced at the end entity HTTP port. The default number is 15. The number you enter in this field is passed to the operating system's `listen()` call.

   Enable. This check box allows you to enable or disable the HTTP port. Uncheck the option if you want to disable the port.

   For issuing certificates to routers (using the CEP protocol), the port must be enabled; see "CEP Enrollment" on page 1092.

SSL port. Type a TCP/IP port number. Certificate Management System uses this port for SSL-enabled communications with the end entity services interface (that is, HTTPS requests from end entities during certificate enrollment, renewal, and revocation). Make sure the port number you specify is unique on the host system.

If you don't want end-entity interaction with a subsystem, for example, if you don't want end entities to interact with a Certificate Manager, you can disable this port too (in addition to the HTTP port). See "Step 6. Enable End-Entity Interaction" on page 405.

Backlog. Type the number of connections that can be waiting to be serviced at the end-entity HTTPS port. The default number is 15. The number you enter in this field is passed to the operating system's `listen()` call.

6. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Step 2: Specify IP Addresses

This step is optional.

You can configure CMS instances to listen to specific IP addresses. For example, you can install the Certificate Manager and Data Recovery Manager on a single host, in separate instances, and then configure the instances so that the Certificate Manager is served on one IP address and the Data Recovery Manager is served on another address.

To clarify this further, consider the machine that hosts the Certificate Manager and Data Recovery Manager has two Ethernet cards that respond to the IP addresses `197.1.137.97` and `197.1.137.98`. You can set up the Certificate Manager to listen to port 443 for the IP address `197.1.137.97` and the Data Recovery Manager to listen to port 443 for the IP address `197.1.137.98`.

To configure a CMS instance to listen to specific IP addresses:

1. Stop the CMS instance; see "Stopping Certificate Management System" on page 137.
2. Open the configuration file in a text editor; to locate the file, see "Locating the Configuration File" on page 85.
3. Add one or more of the following as appropriate:
   - For remote administration port, add this line: `radm.https.host=`
   - For agent port, add this line: `agentGateway.https.host=`
   - For end-entity HTTPS port, add this line: `eeGateway.https.host=`
   - For end-entity HTTP port, add this line: `eeGateway.http.host=`
4. Add the IP address or the host name or interface name as the value for the parameter you just added. For example,
   - If you entered an IP address as the value, the parameter would look similar to this: `radm.https.host=197.1.137.98`
   - If you entered the host name as the value, the parameter would look similar to this: `radm.https.host=cert.netscape.com`
5. If necessary, repeat step 4 for the other ports.
6. Save your changes, and close the configuration file.
7. Start the CMS instance; see "Starting Certificate Management System" on page 128.

Internal Database

Certificate Management System performs various certificate and key-management functions in response to the requests it receives. These functions include the following:

- Storing and retrieving of certificate issuance requests
- Storing and retrieving of certificate records
- Storing of CRLs
- Storing and retrieving of end users' encryption private key records

To fulfill these functions, Certificate Management System maintains a persistent store—a preconfigured Netscape Directory Server—referred to as the *internal database* or *local database*. The internal database is installed automatically as a part of the CMS installation. It is used as an embedded database exclusively by Certificate Management System and can be managed using Directory management tools that come with Netscape Directory Server.

The Directory Server instance used for the internal database is different from the LDAP-compliant directory that you use to manage your corporatewide data (users and groups, their certificates, CRLs, and so on).

- In Netscape Console, you can distinguish an internal database instance from other Directory Server instances. It is in this form:

  `<cms_instance_id>-db`

  `<cms_instance_id>` is the ID of the CMS instance that is using the database. You first specified this when you installed this server.

- If you check the files installed under `<server_root>`, the internal database instance appears like this: `slapd-<cms_instance_id>-db`

Keep in mind that the subsystems use the database for storing different objects. A Certificate Manager stores all the data, certificate issuance requests, certificates, CRLs, and related information; a Registration Manager only stores the certificate issuance requests it receives; and a Data Recovery Manager only stores key records and related data.

Configuring the Internal Database

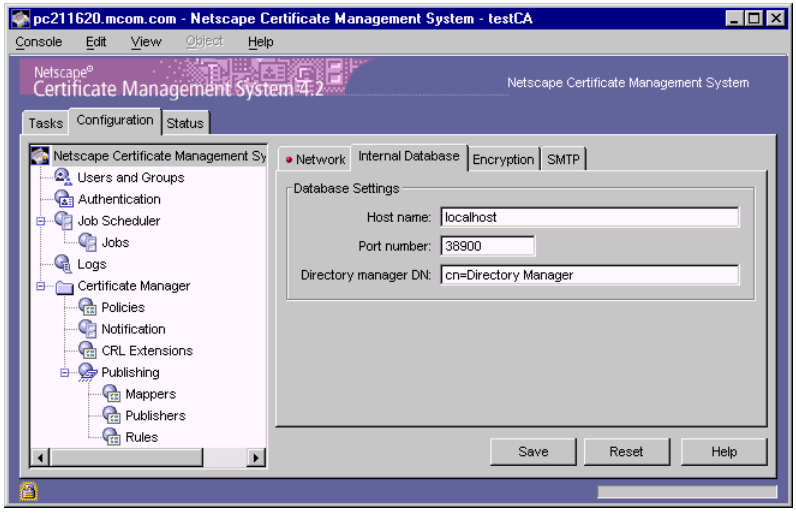
Each instance of Certificate Management System uses a Netscape Directory Server instance as its internal database. All the subsystems that were installed in a CMS instance use the same Directory Server instance to store their data. For example, if you installed a Certificate Manager and Data Recovery Manager together, they use the same internal database for data storage.

Caution The internal database schema is preconfigured for storing CMS data only. Do not make any changes to it or configure Certificate Management System to use any other LDAP directory. Doing so can result in loss of data. Also, do not attempt to use this database for any other purpose.

Step 1. Identify the Directory Server Instance

To identify the Directory Server instance that a CMS instance should use as its internal database:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab, and then in the right pane, select the Internal Database tab.



3. Identify a Directory Server instance by providing the following details:

   Host name. Type the full host name of the machine on which Netscape Directory Server is installed. Certificate Management System uses this name to access the directory. The format for the host name is as follows:

   ```
   <machine_name>.<your_domain>.<domain>
   ```

   By default, the host name of the Directory Server instance being used as the internal database is shown as `localhost` instead of the actual host name (for example, `certificates.netscape.com`). This is done on purpose to insulate the internal database from being visible outside the system—that is, a server on `localhost` can only be accessed from the local machine. Thus, the default configuration minimizes the risk of someone connecting to this Directory Server instance from outside the local machine.

   You can configure the host name to something other than `localhost` if you know what you are doing and you think you can limit the visibility of the internal database to a local subnet. For example, if you installed Certificate Management System and Directory Server on separate machines for load balancing, you will have to specify the host name of the machine in which Directory Server is installed.

   Port number. Type a TCP/IP port number; Certificate Management System uses this port for non-SSL communications with the Directory Server instance that is functioning as the internal database. Make sure that the port you specify is unique on the host system.

   Directory manager DN. Type the distinguished name (DN) of an entry in your LDAP directory that has read and write permission to the entire directory tree. Certificate Management System will use this DN when it accesses the directory tree to communicate with the directory. Keep in mind that the access control set up for this DN determines whether Certificate Management System can communicate with the directory. Typically, you would want to enter the directory manager's DN (the *root DN*) because this DN will have read/write permission to the entire directory tree; see "Root Distinguished Name" on page 1155.

4. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Step 2. Restrict Access to the Internal Database

This step is optional.

Netscape Console displays an entry or icon for the Directory Server instance that Certificate Management System uses as its internal database. You can distinguish an internal database instance from other Directory Server instances. It is in this form: `slapd-<cms_instance_id>-db`

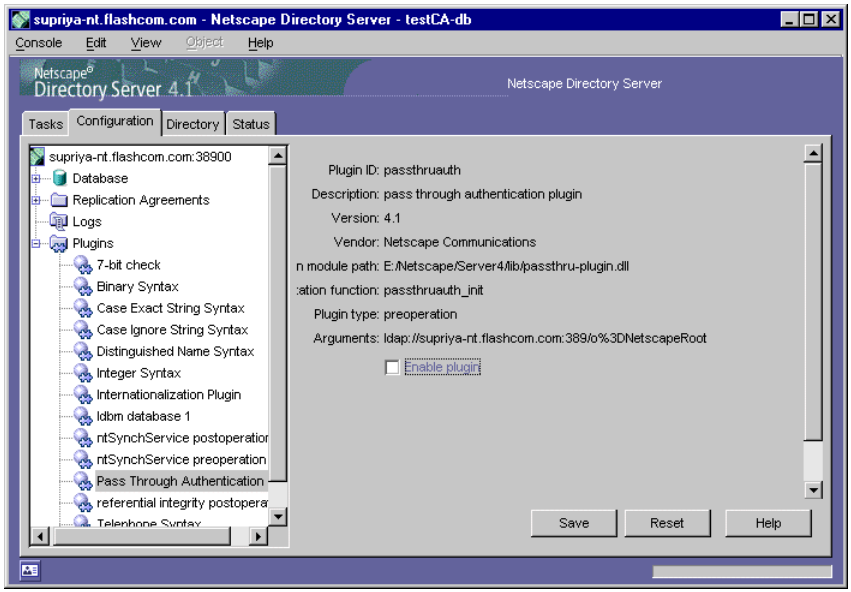
Unlike the CMS window, access to which is restricted to users with *CMS administrator* privileges, the Directory Server window can be accessed by the person who has privileges to access Netscape Console. That is, this person can open the Directory Server window for the internal database and make changes to the data stored there. For example, this person can make changes to the CMS administrators group, such as deleting existing users and adding entries for self.

If you are concerned about this, you can restrict access to the internal database to only those users who know its Directory Manager DN and corresponding password. You can change this password by modifying the single sign-on password cache; see "Changing the Password of an Entry in the Password Cache" on page 149.

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. In the Console tab, select the server group that contains the CMS instance you want.
3. Select the entry that corresponds to the internal database to which you want to restrict access, and click Open.

   The Directory Server window appears.
4. Select the Configuration tab.
5. In the navigation tree, expand Plugins, and then select Pass Through Authentication.

6. In the right pane, uncheck or disable the “Enable plugin” option.



7. Click Save to save your changes.

   You are prompted to restart the server.

8. Click the Tasks tab and click “Restart the Directory Server.”

9. Close the Directory Server window.

10. When the server is restarted, from Netscape Console, open the Directory Server window.

    The “Login to Directory” dialog box appears; the Distinguished Name field displays the Directory Manager DN and you’re required to enter the password that corresponds to this entry.

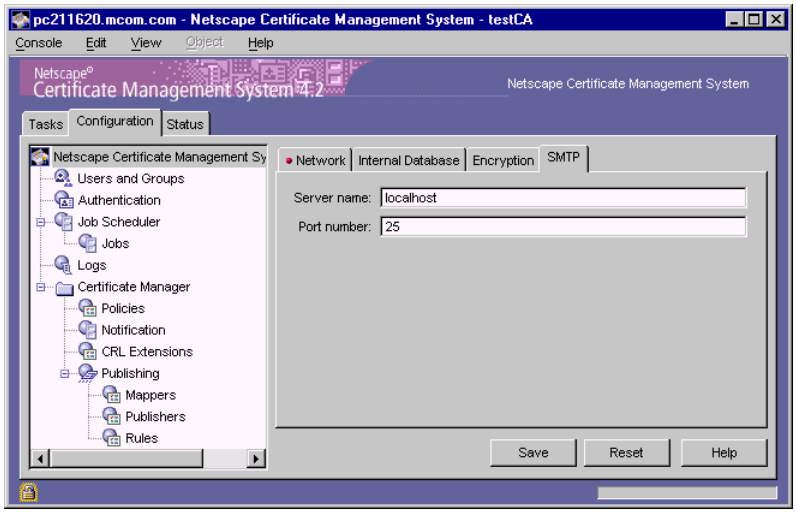
    The Directory Server window (for the internal database) opens only if you enter the correct password.

SMTP Settings

Certificate Management System can send email notifications automatically to users or agents when interesting events occur. For example, you can configure the server to send users email notifications of timed events, such as the expiration of their certificates; for details, see "Job Scheduling and Notification" on page 431.

To identify the mail server that a CMS instance should use for routing email notifications:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab, and then in the right pane, select the SMTP tab.



3. Identify the mail server by providing the following details:

   Server name. Type the full host name of the machine on which your mail server is installed. Certificate Management System uses this name to access the mail server. The format for the host name is as follows:

   `<machine_name>.<your_domain>.<domain>`

   By default, the host name of the mail server is shown as `localhost` instead of the actual host name (for example, `mail.netscape.com`).

Port number. Type the port number at which the mail server is listening for requests.

4. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Chapter

7

Managing Privileged Users and Groups

Privileged users are users who are designated to perform privileged operations on Netscape Certificate Management System (CMS); these operations are privileged because no one else can perform them. You assign *privileged-user* status to a user by storing the user's login information in the internal database of Certificate Management System, associating the user's login information with a personal certificate (if the user is an agent or a trusted manager), and granting access permissions to various CMS resources by adding the user to appropriate groups.

This chapter describes the types of privileged users you need to set up for a CMS instance, what each user does, how Certificate Management System identifies these users, and how you create and manage these users. The chapter also describes what a group is and discusses the groups that Certificate Management System provides by default.

The chapter has the following sections:

Privileged-User Types and Responsibilities

After you install Certificate Management System, your first task is to set up privileged users. There are three types of privileged users: administrators, agents, and trusted managers.

- *Administrators* are users (people) who manage server-specific tasks for the Certificate Manager, Registration Manager, and Data Recovery Manager. For details, see "Administrators" on page 172.
- *Agents* are users (people) who manage the request queues for the Certificate Manager, Registration Manager, and Data Recovery Manager. For details, see "Agents" on page 173.
- *Trusted managers* are CMS subsystems that are connected to other subsystems and that are trusted to perform certain activities for them. For example, you might set up a Registration Manager to screen end-entity certificate requests for a Certificate Manager. Because the Certificate Manager trusts the Registration Manager, it approves all certificate requests received from this Registration Manager. For details, see "Trusted Managers" on page 181.

The role of a privileged user—whether administrator, agent, or trusted manager—is determined by the group to which the user belongs. This is explained in "Groups and Their Privileges" on page 186.

Administrators

Administrators are users who have been assigned CMS administration privileges—permission to access the CMS window and perform all the system administration tasks defined there. You assign these privileges to users by adding them to the internal database and assigning membership in a group called `Administrators` that Certificate Management System creates during installation.

For each CMS instance, the server must have at least one administrator. You can also have more than one individual administering the server.

During installation, Certificate Management System prompts you to provide information for creating the first user entry in the `Administrators` group. Following installation, therefore, this group has a single user entry. For more information about this group, see "Group for Administrators" on page 186.

Certificate Management System authenticates users with administrator-level privileges based on its built-in authentication mechanism. This is explained in "Authentication of Administrators" on page 306.

Agents

Agents are users who have been assigned end-entity certificate- and key-management privileges. Certificate Management System defines three agent roles, one for each of its subsystems: Certificate Manager agents, Registration Manager agents, and Data Recovery Manager agents.

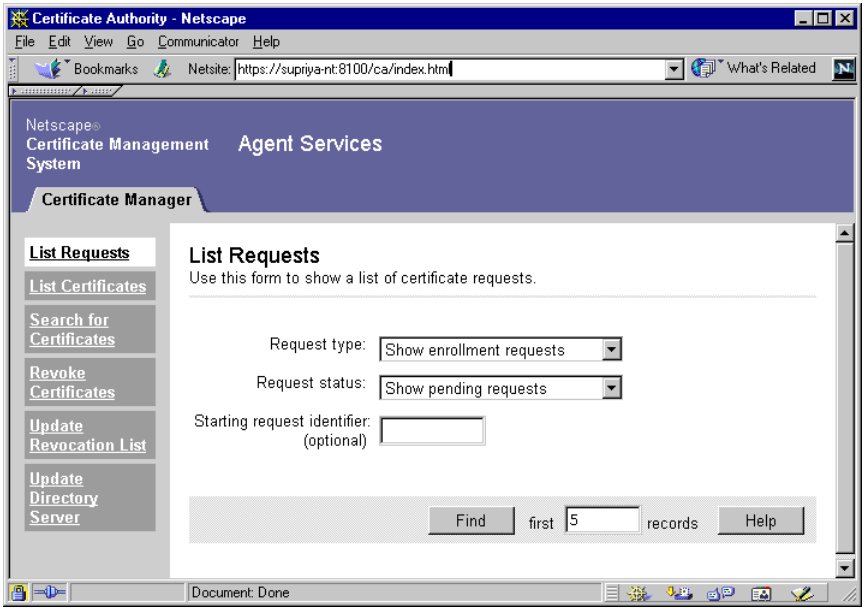
Agents interact with the Certificate Manager, Registration Manager, and Data Recovery Manager to manage operations such as these:

- List, approve, and reject pending certificate issuance and renewal requests
- List certificates
- Search for certificates
- Revoke end-entity certificates
- Manually update certificates and CRLs stored in a publishing directory
- Manage key archival and retrieval requests

All agents perform their tasks through HTML forms-based interfaces. The HTML forms an agent uses to manage a specific subsystem are grouped together and named after the subsystem they represent. For example, the forms-based interface provided for the Certificate Manager is called *Certificate Manager Agent Services* (see Figure 7.1). For more information, see "Agent Services" on page 899.

Agents cannot access the CMS window and perform the tasks provided within the Netscape Console framework—unless they are given administrator privileges.

Figure 7.1 Agents use the HTML forms-based interface called Agent Services



Each subsystem installed in a CMS instance must have at least one agent. You can also have more than one individual managing agent services.

You create agents by adding them to the internal database, assigning membership in the appropriate agent groups, and identifying certificates that the agents must use for SSL client authentication to the subsystem (for it to service requests from the agents). For information about agents' certificates, see "Agent's Certificate for SSL Client Authentication" on page 175. For information on creating agents for a CMS instance, see "Setting Up Agents" on page 193.

During installation, Certificate Management System automatically creates appropriate groups with agent privileges. For more information about these groups, see "Groups for Agents" on page 187.

Certificate Management System authenticates users with agent-level privileges based on its built-in authentication mechanism. This is explained in "Authentication of Agents" on page 308.

Agent's Certificate for SSL Client Authentication

To make a user an agent for a subsystem, one of the things you must do is store the user's client (personal) certificate information in the internal database of the subsystem. For example, if you set up an agent for a Certificate Manager, you store the agent's client certificate in the internal database of that Certificate Manager. Then, when the subsystem receives a request from the agent, it uses this certificate to verify the authenticity of the request before servicing it. For details on how the subsystem verifies the authenticity of a request from an agent, see "Authentication of Agents" on page 308.

If the user you want to set up as an agent does not own a client certificate, ask the user to get one. Depending on your company's PKI policy, the user could get the client certificate from either an internally deployed CA or any public CA.

Keep in mind that the CA that signs your agents' certificates must be *trusted* by the subsystem that processes requests sent by these agents; for example, if your subsystems are set up not to trust public CAs, your agents should not get their certificates signed by public CAs. Make sure that the CA's certificate exists in the subsystem's trust database and that the certificate is valid and trusted. To check whether or not the CA's certificate exists in a subsystem's trust database, follow the instructions in "Viewing the Certificate Database Content" on page 295.

- If the CA's certificate isn't listed, follow the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260 and add the certificate to the subsystem's certificate database.
- If the CA's certificate is listed but *untrusted*, follow the instructions in "Changing the Trust Settings of a CA Certificate" on page 299 and change the setting to *trusted*.

Getting an Agent's Certificate from a Public CA

The following general guidelines explain how a user can get a client certificate from a public CA and how you can copy that certificate (in base-64 encoded form) to the internal database of the appropriate subsystem:

1. The user sends a client certificate request to the public CA from the client machine that he or she will use to access the subsystem from the Agent Services interface. It is important that the user generate and submit this request from the machine she or he will use later to access the subsystem, because part of this request process generates a private key on the local

machine. Alternatively, if location independence is required, the user can use a hardware token, such as a smart card, to generate and store the key pair (and the certificate when the user receives it from the public CA).

2. When the user receives the certificate from the public CA, the user imports the certificate into the web browser that he or she will use to access the subsystem. It is a good idea to ask the user to inform you that the certificate has been installed.

3. Ask the user to send you the certificate information sent by the public CA. In the information that you receive, locate the user's certificate in base-64 encoded form.

   You can also get the user's certificate from the public CA that issued it. Access the public CA site, search for the user's certificate, and locate the certificate in base-64 encoded form.

4. Copy the base-64 encoded certificate, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file.

   The copied information should look similar to the following example:

   ```
   -----BEGIN CERTIFICATE-----
   MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBCMSAwHgYDVQQKExdOZXRzY2FwZSBDb21td
   W5pY2F0aW9uczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFoXDTk5M
   DIyMzE5MDAwMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQ
   LEwZQZW9wbGUxFzAVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0e
   TEjMCEGCSqGSIb3DbndgJARYUc3Vwcml5YUBuZXRzY2FwZS5jb20wXDANBgkqhkiG9w0BAQEFAANL
   ADBIAkEAoYiYgthgtbbnjfngjnjgnagwJjAOBgNVHQ8BAf8EBAMCBLAwFAYJYIZIAYb4QgEBAQHBA
   QDAgCAMA0GCSqGSIb3DQEBBAUAA4GBAFi9FzyJlLmS+kzsue0kTXawbwamGdYql2w4hIBgdR+jWeL
   mD4CP4xzmKdvQ6IqD2q8DBs9lRQu9JYg129oaCLpZfMNTPnMc3WPKO2pWZWUm7waHEtdbo9vSpbJk
   XTM/2GhWbsO5vLdeOxrPGxihkHgV/vmqCl4HW7AorqGgyfygbhgthgutrhryj
   -----END CERTIFICATE-----
   ```

   When you copy the certificate, be sure to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines.

5. Save the text file and use it to store a copy of the certificate in a subsystem's internal database (see "Step 3. Store the Agent's SSL Client Certificate in the Internal Database" on page 198).

Getting an Agent's Certificate from Certificate Management System

The following general instructions explain how a user can get a client certificate from Certificate Management System and how you can copy that certificate (in base-64 encoded form) to the internal database of a subsystem:

1. The user sends a client certificate request to Certificate Management System from the client machine that he or she will use to access the subsystem from the Agent Services interface. It is important that the user generate and submit this request from the machine he or she will use later to access the subsystem, because part of this request process generates a private key on the local machine. Alternatively, if location independence is required, the user can also use a hardware token, such as a smart card, to generate and store the key pair (and the certificate when the user receives it from the public CA).

2. Depending on how your Certificate Management System is configured for certificate issuance, one of the following events happen:

   - If Certificate Management System is configured for manual certification, an issuing agent must process the request and approve it for issuance. Once the request is approved, the server issues the client certificate to the user.

   - If Certificate Management System is configured for automated certification and the request passes authentication and policy checks, the server automatically issues the client certificate to the user.

3. When the user receives the certificate, the user must import the certificate into the web browser that he or she will use to access the subsystem. It is a good idea to ask the user to inform you that the certificate has been installed.

4. After the user imports the certificate into the web browser, you need to copy the certificate (in base-64 encoded form) in order to be able to add it to a subsystem's internal database.

To copy an agent's certificate:

1. Open a web browser window.
2. Go to the Certificate Management System home page for end entities (by default, it is called *End Entity Registration Services*).

   The default URL for this page is in this form:

   `http://<host_name>:<end_entity_HTTP_port>` or

   `https://<host_name>:<end_entity_HTTPS_port>`

   In both cases, the `<host_name>` must be in this form: `<machine_name>.<your_domain>.<domain>`

   For example, the URL may look like this: `https://testCA.siroe.com`
3. Click the Retrieval tab.
4. In the left frame, click either the List Certificates or Search For Certificates link, and search for the user's certificate.
5. In the page listing the results of your search, click the Details button (next to the corresponding user's entry) to see detailed information about the certificate.
6. Scroll down to the section named "Installing This Certificate in a Client," which contains the user's certificate in base-64 encoded form.
7. Copy the base-64 encoded certificate, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file.

   The copied information should look similar to the following example:

   ```
   -----BEGIN CERTIFICATE-----

   MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBCMSAwHgYDVQQKExdOZXRzY2FwZSBDb21td
   W5pY2F0aW9uczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFoXDTk5M
   DIyMzE5MDAwMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQ
   LEwZQZW9wbGUxFzAVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0e
   TEjMCEGCSqGSIb3DbndgJARYUc3Vwcml5YUBuZXRzY2FwZS5jb20wXDANBgkqhkiG9w0BAQEFAANL
   ADBIAkEAoYiYgthgtbbnjfngjnjgnagwJjAOBgNVHQ8BAf8EBAMCBLAwFAYJYIZIAYb4QgEBAQHBA
   QDAgCAMA0GCSqGSIb3DQEBBAUAA4GBAFi9FzyJlLmS+kzsue0kTXawbwamGdYql2w4hIBgdR+jWeL
   mD4CP4xzmKdvQ6IqD2q8DBs9lRQu9JYg129oaCLpZfMNTPnMc3WPKO2pWZWUm7waHEtdbo9vSpbJk
   XTM/2GhWbsO5vLdeOxrPGxihkHgV/vmqCl4HW7AorqGgyfygbhgthgutrhryj

   -----END CERTIFICATE-----
   ```

   When you copy the certificate, be sure to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines.

8. Save the text file and use it to store a copy of the certificate in a subsystem's internal database (see "Step 3. Store the Agent's SSL Client Certificate in the Internal Database" on page 198).

Revocation Status Checking of Agent Certificates

You can also configure the Certificate Manager and Registration Manager to check the revocation status of an agent's certificate the server receives during SSL client authentication; you cannot configure the Data Recovery Manager to check the revocation status of its agents' certificates. The configuration file includes parameters that enable you to specify whether the server should do the revocation checking and if it should, at what interval. Note that the revocation-status verification works for only those agent certificates that have been issued by the Certificate Manager (and not by any third-party CAs).

The set of configuration parameters pertaining to this feature of the Certificate Manager are as follows:

```
auths.revocationChecking.bufferSize=5
auths.revocationChecking.ca=ca
auths.revocationChecking.enabled=true
auths.revocationChecking.unknownStateInterval=0
auths.revocationChecking.validityInterval=120
```

The set of configuration parameters pertaining to this feature of the Registration Manager are as follows:

```
auths.revocationChecking.bufferSize=5
auths.revocationChecking.enabled=true
auths.revocationChecking.ra=ra
auths.revocationChecking.unknownStateInterval=0
auths.revocationChecking.validityInterval=120
```

If you have a Data Recovery Manager installed in the same instance, in addition to the above lines, you'll also notice this line:

```
auths.revocationChecking.kra=kra
```

Table 7.1 provides details for each of the above listed parameters.

Table 7.1 Configuration parameters for checking the revocation status of agents' certificates

Parameter name	Description
`revocationChecking.bufferSize`	Specifies the total number of last-checked certificates the server should maintain in its cache. For example, if you configure the buffer size to be 2, the server retains the last two certificates it checked in its cache. By default, the server caches the last 5 certificates.
`revocationChecking.<subsystem>`	Specifies the name of the CMS instance. `<subsystem>` indicates whether the subsystem is a Certificate Manager (`ca`) or Registration Manager (`ra`). You must not change the default values.
`revocationChecking.enabled`	Specifies whether revocation checking is to be is enabled or disabled. To enable the feature, enter true; to disable the feature, enter false. By default, the feature is enabled.
`revocationChecking.unknownStateInterval`	The default interval is 0 seconds.
`revocationChecking.validityInterval`	Specifies how long, in seconds, the cached certificates are considered valid. Be judicious when choosing the interval, especially when configuring a Registration Manager. For example, if you configure the validity period to be 60 seconds, the server discards the certificates in its cache every minute and attempts to retrieve them from their source—the Certificate Manager uses its internal database to retrieve and verify the revocation status of the certificates, whereas the Registration Manager retrieves certificates from its own internal database and then requests the Certificate Manager for the revocation status of these certificates. The default validity period is 120 seconds (2 minutes).

To configure a Certificate Manager or Registration Manager to verify the revocation status of its agents' certificates:

1. Stop the CMS instance; see "Stopping Certificate Management System" on page 137.
2. Open the configuration file in a text editor; to locate the file, see "Locating the Configuration File" on page 85.
3. Locate the parameters mentioned above and edit their values as appropriate.
4. Save your changes, and close the configuration file.
5. Start the CMS instance; see "Starting Certificate Management System" on page 128.

Trusted Managers

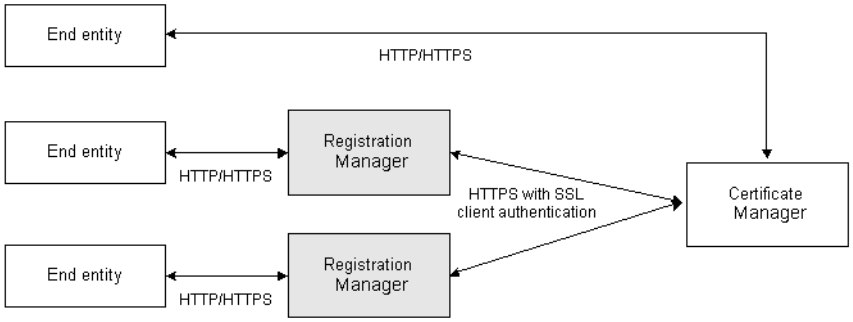
Trusted managers are those CMS subsystems that are connected to other CMS subsystems and that are trusted to perform specific functions for them. In other words, a trusted manager acts as a front end to the subsystem that trusts it, performing specific functions, depending on the subsystem to which it is connected. You establish this trust between the two subsystems by configuring them to function in certain way.

Subsystems That Can Function as Trusted Managers

In Certificate Management System, the Registration Manager or Certificate Manager can function as a trusted manager; the Data Recovery Manager cannot function as a trusted manager. You can configure

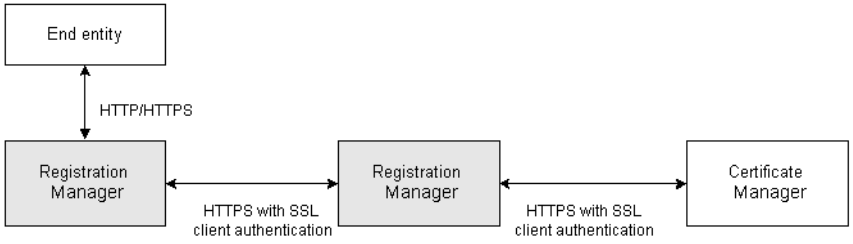
- A Certificate Manager to delegate its end-entity interactions to a trusted Registration Manager, for reasons of localizability (proximity to end entities), customizability, and CA scalability; the Certificate Manager trusts the Registration Manager and signs all certificate signing requests sent by this Registration Manager.

For example, as illustrated in the figure below, you might deploy one or more Registration Managers to process, approve, and forward certificate signing requests to a Certificate Manager.



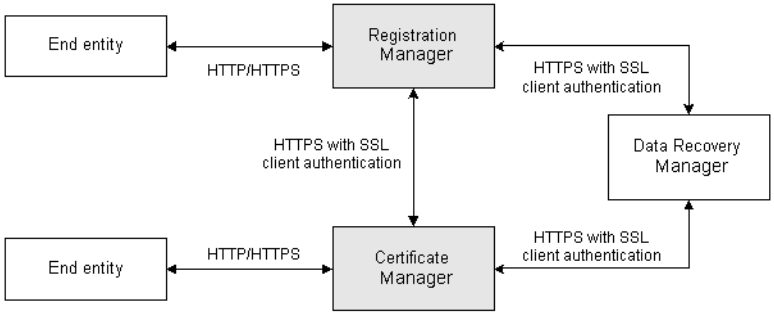
- A Registration Manager to delegate its end-entity interactions to a trusted Registration Manager, for reasons of localizability (proximity to end entities), customizability, and scalability; the Registration Manager trusts the Registration Manager and services all certificate requests sent by this Registration Manager.

  For example, as illustrated in the figure below, you might deploy one or more Registration Managers to forward requests to another Registration Manager in a Registration Manager chain. (The Registration Manager passing the request acts as the client and the one receiving the request acts as the server.)



- A Data Recovery Manager to delegate its end-entity interactions to a trusted Certificate Manager or Registration Manager for security reasons; the Data Recovery Manager trusts the Certificate Manager or Registration Manager and services all key archival and recovery requests initiated by this subsystem.

  For example, as illustrated in figure below, you might deploy one or more Certificate Managers or Registration Managers to send key archival or recovery requests to a Data Recovery Manager.



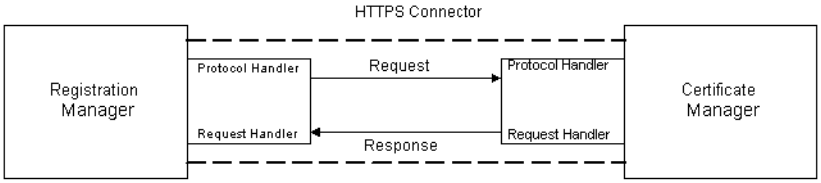
Connectors for Linking Trusted Managers

Certificate Management System supports proprietary HTTPS connectors for linking CMS subsystems. You can use these connectors to make the following connections:

- Registration Manager to Certificate Manager
- Registration Manager to Registration Manager
- Registration Manager to Data Recovery Manager
- Certificate Manager to Data Recovery Manager

Figure 7.2 illustrates how a trusted Registration Manager communicates with a Certificate Manager.

Figure 7.2 Connectivity service between a trusted Registration Manager and other subsystems



Keep in mind that a trusted manager does not take on the main functions of the subsystem that trusts it. For example, if a Registration Manager is connected to a Certificate Manager, the Registration Manager has no authority to issue (sign) certificates or CRLs. It receives end-entity requests, authenticates them, and forwards them to the Certificate Manager for signing. After receiving a response from the Certificate Manager, it notifies the end entity of the results.

Similarly, a Certificate Manager or Registration Manager connected to a Data Recovery Manager has no authority to archive and recover end users' encryption private keys.

You can configure a subsystem to trust one or more managers. You do this by adding these managers as privileged users to the internal database of that subsystem, assigning them memberships in the appropriate group, and identifying the certificates the managers must use for SSL client authentication to the subsystem they report to. For information about adding a trusted manager, see "Setting Up Trusted Managers" on page 201.

During installation, Certificate Management System automatically creates a group with trusted manager privileges. For more information about this group, see "Group for Trusted Managers" on page 189.

Trusted Manager's Certificate for SSL Client Authentication

By default, a Registration Manager that has been set up to function as a trusted manager uses its *signing certificate* for SSL client authentication to the subsystem that trusts it. For information on this certificate, see "Signing Key Pair and Certificate" on page 230. Similarly, a Certificate Manager that has been set

up to function as a trusted manager uses its *SSL server certificate* for SSL client authentication to the subsystem that trusts it. For information on this certificate, see "SSL Server Key Pair and Certificate" on page 228.

When you set up a trusted manager for a CMS subsystem, it is important to know which CA has issued the certificate the trusted manager will use for SSL client authentication to the subsystem. The certificate must be issued by a CA that the subsystem trusts. For example, when you set up a trusted Registration Manager for a subsystem, it is important to know which CA has issued the Registration Manager's signing certificate. The certificate must be issued by a CA that the subsystem trusts. If the subsystem is a Certificate Manager, the certificate must be issued by either the Certificate Manager itself or a CA that the Certificate Manager trusts. Similarly, if the Registration Manager is connected to a Data Recovery Manager, the signing certificate must be issued by the CA that the Data Recovery Manager trusts.

The issuer of a Registration Manager's signing certificate is the CA from which you requested the certificate when you installed the Registration Manager. If you have renewed the certificate since installation, the issuer is the CA from which you requested the renewed certificate. Check the signing certificate for its issuer's name; see "Viewing the Certificate Database Content" on page 295. You can also find this information by looking at the installation worksheet you completed in preparation for installing the system.

Once you learn the issuer's name, verify that this CA's certificate exists in the subsystem's trust database and that the certificate is trusted. To check whether the CA's certificate exists in the subsystem's trust database, follow the instructions in "Viewing the Certificate Database Content" on page 295.

- If the CA's certificate isn't listed, follow the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260 and add the certificate to the subsystem's certificate database.
- If the CA's certificate is listed but *untrusted*, follow the instructions in "Changing the Trust Settings of a CA Certificate" on page 299 and change the trust setting to *trusted*.

Groups and Their Privileges

In Certificate Management System, a *group* refers to a collection of privileged users—administrators, agents, or trusted Registration Managers. Each group has predetermined privileges, based on its access control. All users belonging to a group automatically inherit the privileges of that group.

When you installed Certificate Management System, it automatically created the following groups for the subsystems you installed:

- Group for Administrators
- Groups for Agents
- Group for Trusted Managers

These default groups are created in the internal database of the appropriate CMS instance. They can help you set up your privileged users quickly and easily.

Do not delete or change the group names. Also, don't change the internal database in which the groups are stored. However, you can add new privileged users to these groups; see “Setting Up Privileged Users” on page 190.

Group for Administrators

During installation, Certificate Management System automatically creates a group called `Administrators` and adds a user to this group; the server sets the name of this user to the *certificate administrator ID* (of the CMS administrator) you specified during installation. If you don’t remember this name, see the installation worksheet you completed in preparation for installing the system. For example, if you specified `admin` as the user ID for the CMS administrator, the name of the user in the `Administrators` group will be `admin`.

Keep in mind that the `Administrators` group must always contain at least one user entry. This means you can delete the entry that was created in this group during installation, provided you add another user to the group.

After installation, you must do the following:

1. Log in to the CMS window with the administrator ID and password specified during installation.

2. Depending on the components you installed, create one or more privileged users and add them to the appropriate groups. It is recommended that you add at least one more user to the `Administrators` group. For instructions on creating privileged users and adding them to one or more groups, see "Setting Up Privileged Users" on page 190.

Groups for Agents

Depending on the subsystems you chose to install, Certificate Management System automatically creates a combination of the following groups for a CMS instance:

- `Certificate Manager Agents` group, if you have installed the Certificate Manager
- `Registration Manager Agents` group, if you have installed the Registration Manager
- `Data Recovery Manager Agents` group, if you have installed the Data Recovery Manager

Group for Certificate Manager Agents

When the Certificate Manager is installed, a group called `Certificate Manager Agents` is automatically created in its internal database. After installation, this group has a single user entry—when you get the first agent certificate from the Certificate Manager, the server automatically adds the initial administrator as the agent and stores a copy of the agent certificate against that user entry. The user ID for this agent user is the same as the *certificate administrator ID* (as specified during installation).

The `Certificate Manager Agents` group has access rights to agent-specific resources of the Certificate Manager; that is, privileged users you add to this group automatically inherit access rights to the agent port of the Certificate Manager. For information on ports, see "CMS Ports" on page 155.

After installation, you should add to this group the privileged users to whom you want to assign Certificate Manager agent privileges. All agents who belong to the `Certificate Manager Agents` group can access the Certificate Manager Agent Services interface; see "Certificate Manager Agent Services" on page 900.

For an agent to be able to carry on SSL client-authenticated communication with a Certificate Manager, you need to do additional configurations. See "Setting Up Agents" on page 193.

Group for Registration Manager Agents

When the Registration Manager is installed, a group called `Registration Manager Agents` is automatically created in its internal database. By default, this group has no entries.

The `Registration Manager Agents` group has access rights to agent-specific resources of the Registration Manager; that is, privileged users you add to this group automatically inherit access rights to the agent ports of the Registration Manager. For information on ports, see "CMS Ports" on page 155.

After installation, you should add to this group the privileged users to whom you want to assign Registration Manager agent privileges. All agents who belong to the `Registration Manager Agents` group can access the Registration Manager Agent Services interface; see "Registration Manager Agent Services" on page 901.

For an agent to be able to do SSL client-authenticated communication with a Registration Manager, you need to do additional configurations. See "Setting Up Agents" on page 193.

Group for Data Recovery Manager Agents

When the Data Recovery Manager is installed, a group called `Data Recovery Manager Agents` is automatically created in its internal database. By default, this group has no entries. Note that if the Data Recovery Manager is colocated with a Certificate Manager, following installation, this group has a single user entry—when you get the very first agent certificate from the Certificate Manager, the server automatically adds the initial administrator as the agent and stores a copy of the agent certificate against that user entry. The user ID for this agent user is the same as the *certificate administrator ID* (as specified during installation).

The `Data Recovery Manager Agents` group has access rights to agent-specific resources of the Data Recovery Manager; that is, privileged users you add to this group automatically inherit access rights to the agent ports of the Data Recovery Manager. For information on ports, see "CMS Ports" on page 155.

After installation, you should add to this group the privileged users to whom you want to assign Data Recovery Manager agent privileges. All agents who belong to the `Data Recovery Manager Agents` group can access the Data Recovery Manager Agent Services interface; see "Data Recovery Manager Agent Services" on page 902.

For an agent to be able to carry on SSL client-authenticated communication with a Data Recovery Manager, you need to do additional configurations. See "Setting Up Agents" on page 193.

Group for Trusted Managers

When the Certificate Manager, Registration Manager, or Data Recovery Manager is installed, a group called `Trusted Managers` is automatically created in its internal database. By default, this group has no entries.

The `Trusted Managers` group has access rights to the corresponding agent gateway; that is, the subsystems you add to this group automatically inherit access rights to the agent port of the corresponding Certificate Manager, Registration Manager, or Data Recovery Manager. For information on ports, see "CMS Ports" on page 155.

After installation, you should add to this group the subsystems that you want to function as trusted managers. All subsystems that belong to the `Trusted Managers` group can carry on SSL client-authenticated communication with the subsystem that trusts them and receive responses back.

For a Registration Manager to be able to do SSL client-authenticated communication with a subsystem, you need to do additional configurations. See "Setting Up Trusted Managers" on page 201.

Setting Up Privileged Users

Setting up privileged users for a CMS instance involves adding the appropriate user information to the internal database of that instance. You can set up any number of privileged users for a CMS instance. If the user is a person (that is, an administrator or agent), you can put that user into as many groups as you like.

This section describes the following tasks:

- Setting Up Administrators
- Setting Up Agents
- Setting Up Trusted Managers

Setting Up Administrators

You need at least one administrator for each instance of Certificate Management System. To understand the role of an administrator, see "Administrators" on page 172.

This section explains how to add administrators to a CMS instance manually. To import users with administrator-level privileges from a Netscape Certificate Server version 1.0x database into the internal database used by a CMS instance, see "Appendix A, Migrating from Certificate Server" in the *Netscape Certificate Management System Deployment and Installation Guide*.

Setting up an administrator involves the following steps:

- Step 1. Find the Required Information
- Step 2. Add the Information to the Internal Database

Step 1. Find the Required Information

Note the user's corporate information, such as name, user ID, email address, and phone number.

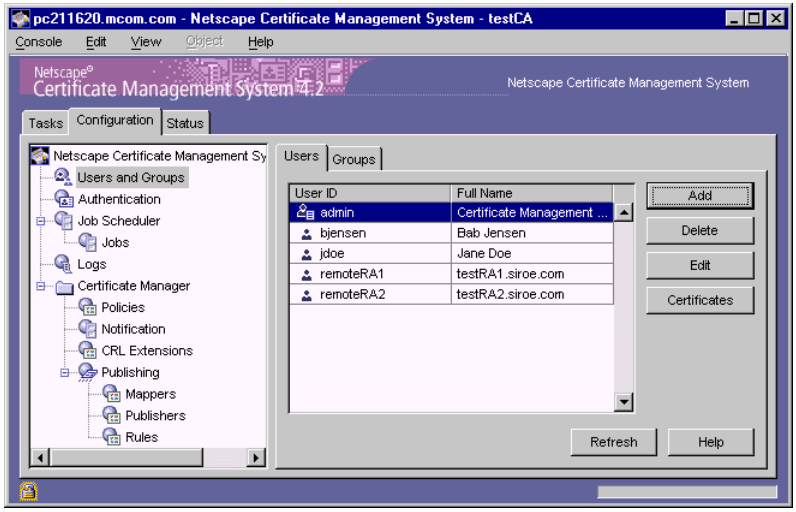
Step 2. Add the Information to the Internal Database

To add the information to the internal database of a CMS instance:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

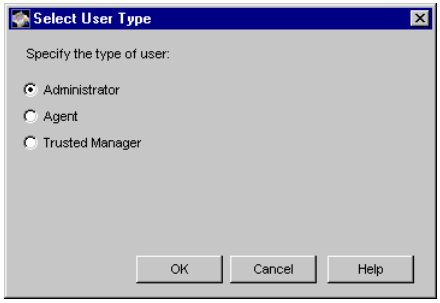
2. In the navigation tree, select Users and Groups.

   The Users tab appears on the right pane.



3. Click Add.

   The Select User Type window appears.



4. Select Administrator and click OK.

The Edit User Information window appears.

5. Specify information as appropriate:

   User ID. Type a user ID or login name for the user. The ID can be an alphanumeric string of up to 255 characters. Give this ID to the user. The user is required to enter this ID in the login screen of the CMS window; see "Logging In to the CMS Window" on page 78.

   Full name. Type the user's full name. The user never sees this. This field is to help you keep track of your users. The name can be an alphanumeric string of up to 255 characters.

   Password. Type a password of up to eight characters for the user. Give this password to the user. The user is required to enter this password in the login screen of the CMS window; see "Logging In to the CMS Window" on page 78.

   Confirm password. Retype the password exactly as you typed it in the Password field.

   Email. Type the user's complete email address. The user never sees this. This field is to help you contact the user, if the need arises.

   Phone. Type the user's phone number. The user never sees this. This field is to help you contact the user, if the need arises.

   Group. Select Administrators; for more information about this group, see "Group for Administrators" on page 186. When you set up a user, you can add him or her to only one group. To add the user to another group, see "Changing Members in a Group" on page 221.

6. Click OK.

You are returned to the Users tab. The administrator you just added will be displayed in the list of users.

7. Click Refresh to view the updated configuration.

Setting Up Agents

You need an agent for each subsystem installed in a given CMS instance. To understand the role of an agent, see “Agents” on page 173. This section explains how to add agents to a CMS instance. To import users with agent-level privileges from a Netscape Certificate Server version 1.0x database, see "Appendix A, Migrating from Certificate Server" in the *Netscape Certificate Management System Deployment and Installation Guide*.

You can set up agents for a CMS instance in two ways:

- Setting up Agents Using the Automated Process
- Setting up Agents Using the Manual Process

Setting up Agents Using the Automated Process

Certificate Management System automates the process of setting up agents if agents request their certificate using the *manual* enrollment form. The automated process is built into the request-approval form (the page that displays the pending request) in the Agent Services interface and it enables the person who has both *Certificate Manager agent* and *Administrator* privileges to create new agents for a CMS instance—that is, the Certificate Manager agent who approves new agents' certificate requests must belong to both `Certificate Manager Agents` and `Administrators` groups in the internal database of the Certificate Manager.

The request-approval form includes a checkbox labeled “This certificate is for a <subsystem> agent”, where <subsystem> indicates Certificate Manager, Registration Manager, or Data Recovery Manager. Selecting the checkbox indicates that the user who has requested the certificate should be made an agent for the specified subsystem. Selecting the checkbox also requires the Certificate Manager agent to specify a user ID for the new agent.

If the Certificate Manager agent approves the certificate request with the checkbox selected and user ID specified, the server automatically adds the user as an agent to its internal database, copies the user's client certificate to the database, and associates the certificate with the new user's entry.

If you want to test this feature, follow these steps:

1. Open a web browser window.
2. Access the end-entity interface.
3. In the Enrollment tab, under Browser, select Manual.
4. In the enrollment form that appears, enter sample data and submit the request.
5. Next, access the Certificate Manager Agent Services interface.
6. Click List Requests.
7. In the page that displays, select the "Show pending requests", and click Find.
8. In the list of certificate signing requests that displays, select the request you submitted.
9. In the request approval form for user enrollment requests, verify the request. If required, adjust some of the parameters such as the subject name and validity period.
10. Next, check the box labeled "This certificate is for a Certificate Manager agent", specify a user ID for the new agent, and approve the certificate request.

    The Certificate Manager processes the requests, issues the certificate to the user, automatically adds the user as an agent to its user and group database, copies the user's client certificate to the database, and associates the certificate with the new user's entry.
11. To verify, log in to the CMS window for the Certificate Manager.
12. In the navigation tree, click Users and Groups.

13. In the list of users, you should find the user ID you specified for the new agent. To view the certificate issued to the new agent, select the user ID and click Certificates.

Setting up Agents Using the Manual Process

Typically, you add agents to a CMS instance by manually creating a privileged-user entry for the agent in the corresponding subsystem's user and group database and then add the agent's certificate to the database.

Setting up an agent involves the following steps:

- Step 1. Find the Required Information
- Step 2. Add the Information to the Internal Database
- Step 3. Store the Agent's SSL Client Certificate in the Internal Database
- Step 4. Check the Certificate Database for the CA Certificate

Step 1. Find the Required Information

Before adding an agent to the internal database of a CMS instance:

- Note the user's corporate information, such as name, login ID, password, email address, and phone number.
- Make sure the user has one or more client certificates that are currently valid; the certificate must not have expired, been revoked, or been signed by an *untrusted* authority. If the user does not own a client certificate, either issue the user a certificate or ask the user to get a certificate. For details, see "Agent's Certificate for SSL Client Authentication" on page 175.
- Identify the certificate that the user must use for SSL client authentication to Certificate Management System. You can identify more than one certificate if you want.
- Copy this certificate, in base-64 encoded format, to a text file.

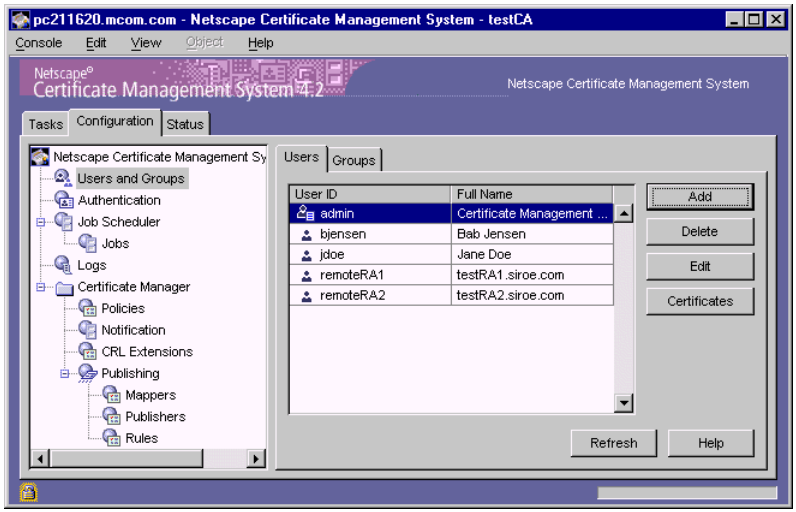
Step 2. Add the Information to the Internal Database

To add the information to the internal database of a CMS instance:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

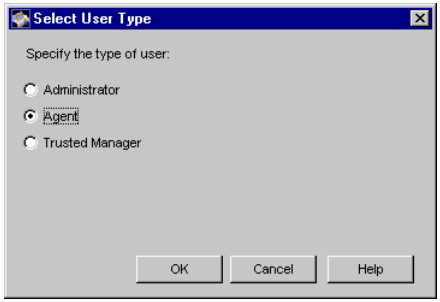
2. In the navigation tree, select Users and Groups.

   The Users tab appears.



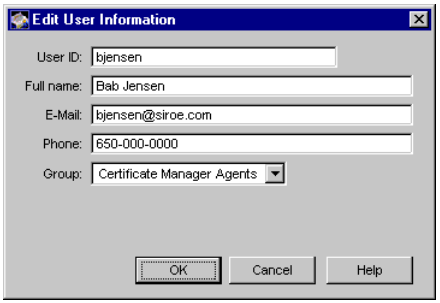
3. Click Add.

   The Select User Type window appears.



4. Select Agent and click OK.

The Edit User Information window appears.



5. Specify information as appropriate.

   The information you enter here is to help you keep track of your agent users; the user never sees or uses it. The server relies solely on the agent's client certificate (which you will add next) for authentication.

   User ID. Type the user ID or login name. The ID can be an alphanumeric string of up to 255 characters.

   Full name. Type the user's full name. The name can be an alphanumeric string of up to 255 characters.

   Email. Type the user's complete email address.

   Phone. Type the user's phone number.

   Group. Choose the appropriate agent group; for more information about this group, see "Groups for Agents" on page 187. When you set up a user, you can add her or him to only one group. To add the user to another group, see "Changing Members in a Group" on page 221.

6. Click OK.

   You are returned to the Users tab. The agent you just added is displayed in the list of users.

What you do next depends on whether you have the agent's certificate:

- If you copied the user's certificate in base-64 encoded form to a text file, proceed to Step 3. (For details on getting the user's certificate, see "Agent's Certificate for SSL Client Authentication" on page 175.)
- Otherwise, save your changes.

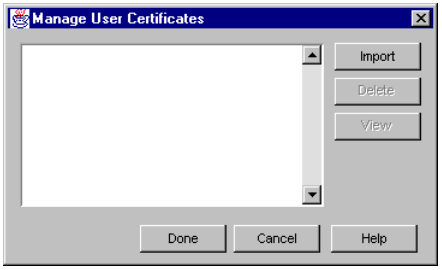
You can add the certificate to the internal database later, following the instructions provided in “Changing a Privileged User’s Certificate” on page 220.

Step 3. Store the Agent’s SSL Client Certificate in the Internal Database

To store a copy of an agent’s SSL client certificate in the internal database:

1. In the Users tab, click Certificates.

   The Manage User Certificates window appears.

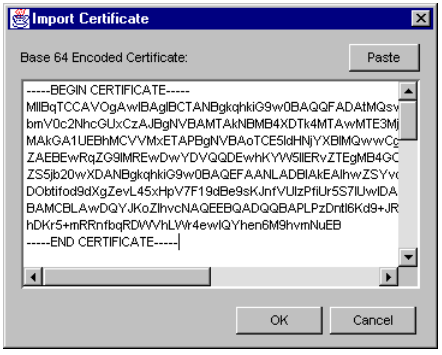


2. Click Import.

   The Import Certificate window appears.

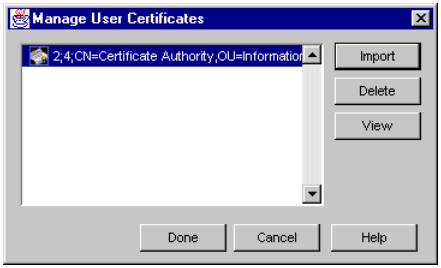
3. Click inside the text area, and paste the user’s certificate in base-64 encoded form.

   Be sure to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines.



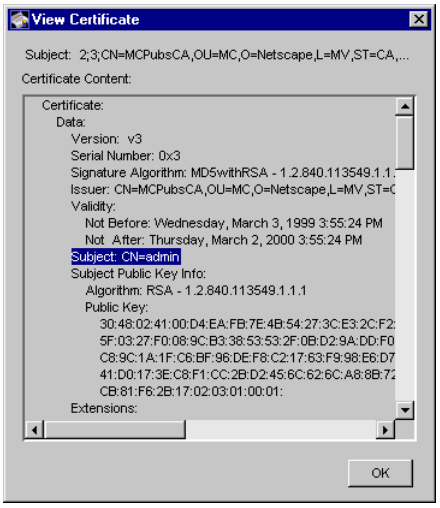
4. Click OK.

You are returned to the Manage User Certificates window. The certificate you imported should now be listed in this window.



5. To view the certificate you imported, select it and click View.

   The certificate information appears.



6. Click Done.

   You are returned to the Users tab.

7. Click Refresh to view the updated configuration.

Step 4. Check the Certificate Database for the CA Certificate

The CA that signed the agent's SSL client certificate must be *trusted* by the subsystem that services requests from the agent. Make sure that this CA's certificate exists in the subsystem's certificate database (internal or external)

and that it is trusted. To check whether the CA's certificate exists in your subsystem's certificate database, follow the instructions in "Viewing the Certificate Database Content" on page 295.

- If the CA certificate isn't listed, follow the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260 and add the certificate to the certificate database.
- If the CA's certificate is listed but *untrusted*, follow the instructions in "Changing the Trust Settings of a CA Certificate" on page 299 and change the trust setting to *trusted*.

Setting Up Trusted Managers

You can set up a Registration Manager or Certificate Manager to function as a trusted manager to another CMS instance. This section explains how to do this.

- Setting up Trusted Managers Using the Automated Process
- Setting Up a Registration Manager as a Trusted Manager
- Setting Up a Certificate Manager as a Trusted Manager

To understand the role of a trusted manager in your PKI, see "Trusted Managers" on page 181.

Setting up Trusted Managers Using the Automated Process

Certificate Management System automates the process of setting up trusted managers. The automated process is built into the request-approval form (the page that displays the pending request) in the Agent Services interface and it enables the person who has both *Certificate Manager agent* and *Administrator* privileges to create new trusted managers for a CMS instance—that is, the Certificate Manager agent who approves the subsystems' certificate requests must belong to both the Certificate Manager Agents and Administrators groups in the user and group database of the Certificate Manager. For more information about these groups, see "Groups and Their Privileges" on page 186.

- The request-approval form for Certificate Manager's SSL server certificate request includes a checkbox labeled "This certificate is for a Trusted Manager."
- Similarly, The request-approval form for Registration Manager's signing certificate request includes a checkbox labeled "This certificate is for a Trusted Manager."

If selected, the checkbox indicates that the subsystem that has requested the certificate must be made a trusted manager. Selecting the checkbox also requires the agent to specify an ID for the subsystem that will be set up as a trusted manager.

If the Certificate Manager agent approves the certificate request with the checkbox selected and user ID specified, the server automatically adds the subsystem as a new privileged user to its user and group database, adds the user to the Trusted Managers group, copies the corresponding certificate to the database, and associates the certificate with the new user's entry.

Note that for a Certificate Manager to add the Registration Manager this way, the Certificate Manager agent who approves the Registration Manager signing certificate request must belong to both the Certificate Manager Agents and Administrators groups in the internal database of the Certificate Manager. For more information about these groups, see "Groups and Their Privileges" on page 186.

Setting Up a Registration Manager as a Trusted Manager

You can set up a remote Registration Manager to function as a trusted manager to a Certificate Manager, another Registration Manager, or a Data Recovery Manager.

- Step 1. Find the Required Information
- Step 2. Create a User Entry for the Registration Manager
- Step 3. Copy the Registration Manager's Certificate to the Internal Database
- Step 4. Check the Certificate Database for the CA Certificate
- Step 5. Configure Registration Manager's Connector Settings

Step 1. Find the Required Information

Before setting up a Registration Manager to function as a trusted manager to another CMS subsystem:

- Note identifying information, such as the instance ID and host name of the Registration Manager.
- Make sure that the Registration Manager has the certificate you want it to use for SSL client authentication to the subsystem that will trust it; by default, the Registration Manager uses its *signing certificate* for this purpose. The certificate must be currently valid; the certificate must not have expired, been revoked, or been signed by an authority *untrusted* by the subsystem. For details, see "Trusted Manager's Certificate for SSL Client Authentication" on page 184.
- Locate the certificate in base-64 encoded format. Copy the certificate, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file.
- Identify the subsystem—Certificate Manager, Registration Manager, or Data Recovery Manager—to which you want to connect the Registration Manager. Note details, such as the host name and port number of that subsystem.
- If you are planning to connect the Registration Manager to a Certificate Manager, keep this in mind: during the installation of a Registration Manager, you generated a signing certificate for the Registration Manager. If you requested the signing certificate from a Certificate Manager, you were given an opportunity to add the Registration Manager as a trusted manager to that Certificate Manager's database. If you chose this option, then the Registration Manager is already set up to function as a trusted manager to that Certificate Manager—in this case, you are not required to go through these steps.

Step 2. Create a User Entry for the Registration Manager

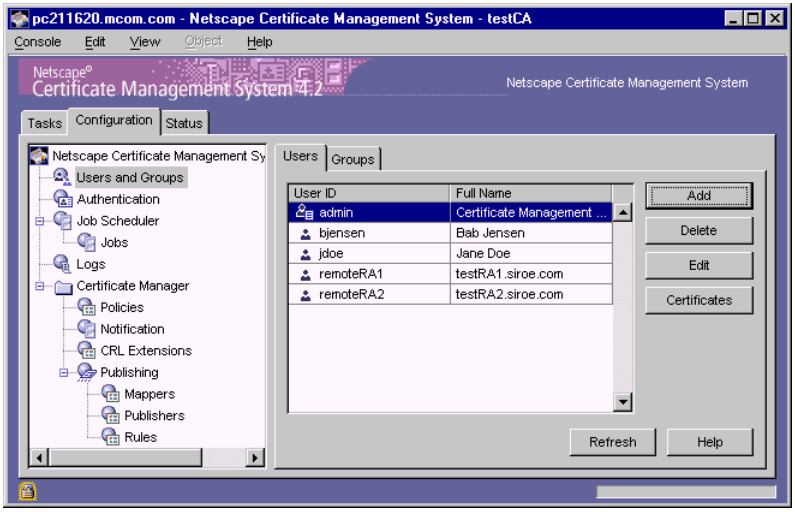
In this step, you create a privileged-user entry for the Registration Manager in the internal database of the subsystem. As a part of creating this entry, you also add the user entry to the `Trusted Managers` group in order to give the entry access privileges to the agent port of the subsystem.

To create a user entry with appropriate access privileges for a Registration Manager:

1. Log in to the CMS window for the subsystem (see "Logging In to the CMS Window" on page 78). For the purposes of completing these instructions, let us assume that the subsystem is a Certificate Manager.

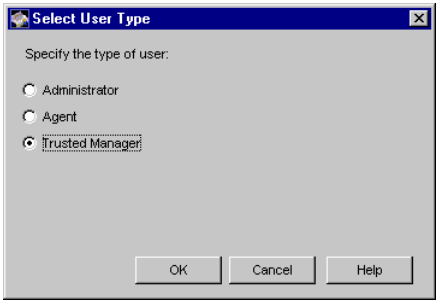
2. In the navigation tree, select Users and Groups.

   The Users tab appears.



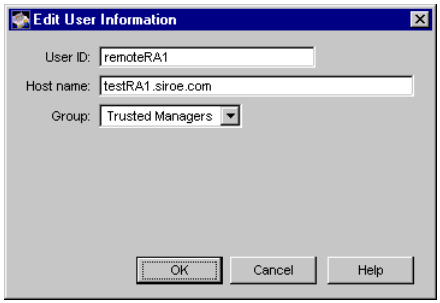
3. Click Add.

   The Select User Type window appears.



4. Select Trusted Manager and click OK.

The Edit User Information window appears.



5. Specify information as appropriate.

   The information you enter here is to help you keep track of the Registration Manager; the subsystem never uses it. The subsystem relies solely on the Registration Manager's SSL client certificate (which you will add in Step 3) for authentication.

   User ID. Type the Registration Manager's instance ID (or any other ID that will help you identify the Registration Manager in the list of privileged users). The ID can be an alphanumeric string of up to 255 characters.

   Host name. Type the full host name of the Registration Manager. The host name can be an alphanumeric string of up to 255 characters. It must be in the `<machine_name>.<your_domain>.<domain>` form.

   Group. Select Trusted Managers; for more information about this group, see "Group for Trusted Managers" on page 189.

6. Click OK.

   You are returned to the Users tab. The Registration Manager you just added appears in the list of users.

What you do next depends on whether you have the Registration Manager's SSL client certificate:

- If you copied the Registration Manager's certificate in base-64 encoded form to a text file, proceed to Step 3. (For details on getting this certificate, see "Trusted Manager's Certificate for SSL Client Authentication" on page 184.)
- Otherwise, skip to Step 5. You can add the certificate later, following the instructions in "Changing a Privileged User's Certificate" on page 220.

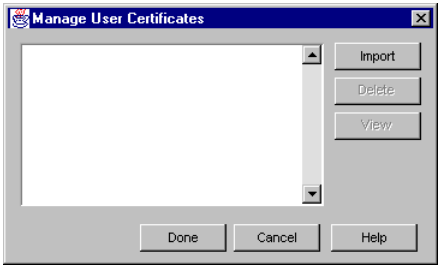
Step 3. Copy the Registration Manager's Certificate to the Internal Database

In this step, you add a copy of the Registration Manager's SSL client authentication certificate to the internal database of the subsystem and associate the certificate with the user entry you created in Step 2.

To store the Registration Manager's SSL client certificate in the internal database of the subsystem:

1. In the Users tab, select the user entry you just added for the Registration Manager and click Certificates.

   The Manage User Certificates window appears.

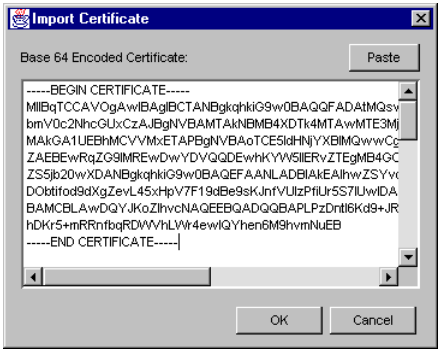


2. Click Import.

   The Import Certificate window appears.

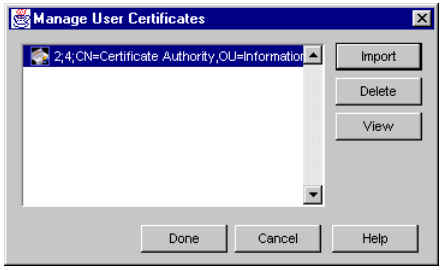
3. Click inside the text area, and paste the Registration Manager's certificate in base-64 encoded form.

   Be sure to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines.



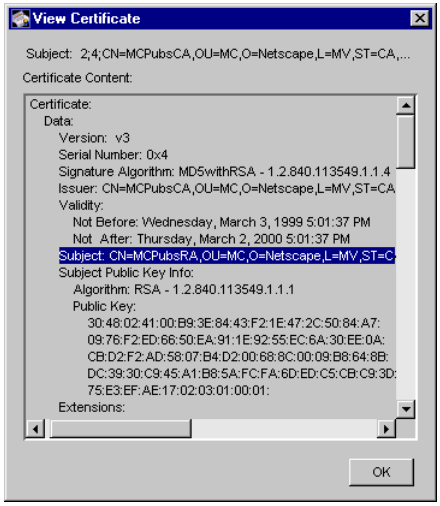
4. Click OK.

   You are returned to the Manage User Certificates window. The certificate you imported should now be listed in this window.



5. To view the certificate you imported, select it and click View.

   The certificate information appears. Verify that the certificate you added is the correct one.



6. Click Done.

   You are returned to the Users tab.

Step 4. Check the Certificate Database for the CA Certificate

The issuer of the Registration Manager's certificate that you added in Step 3 must be *trusted* by the subsystem that services certificate requests approved by the Registration Manager. Make sure that this CA's certificate exists in the

subsystem's certificate database (internal) and that it is trusted. To check whether the CA's certificate exists in the subsystem's certificate database, follow the instructions in "Viewing the Certificate Database Content" on page 295.

- If the CA certificate isn't listed, follow the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260 and add the certificate to the certificate database.
- If the CA's certificate is listed but *untrusted*, follow the instructions in "Changing the Trust Settings of a CA Certificate" on page 299 and change the trust setting to *trusted*.

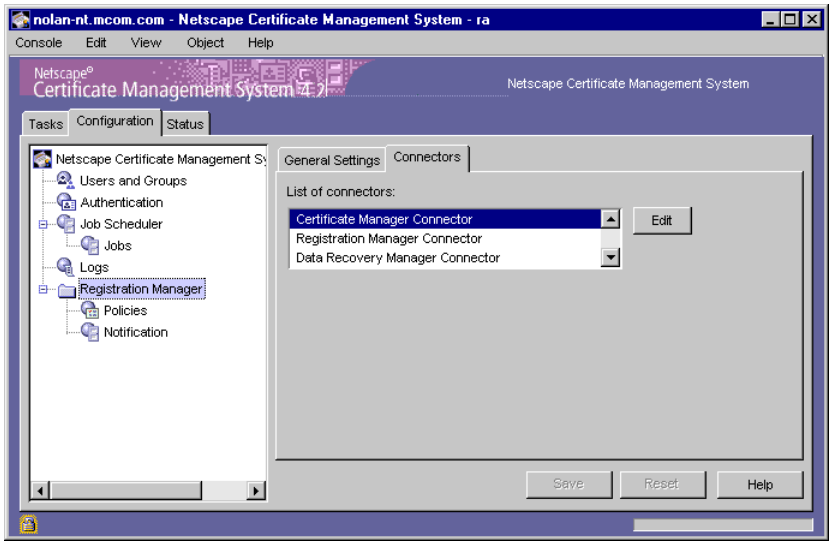
Step 5. Configure Registration Manager's Connector Settings

In this step, you configure the connector settings of the Registration Manager. This enables the Registration Manager to utilize the proprietary HTTPS connectors to communicate with the subsystem (following successful SSL client authentication).

1. Log in to the CMS window for the Registration Manager (see "Logging In to the CMS Window" on page 78).
2. In the navigation tree, select Registration Manager.

   The General Settings tab appears in the right pane.

3. Select the Connectors tab.

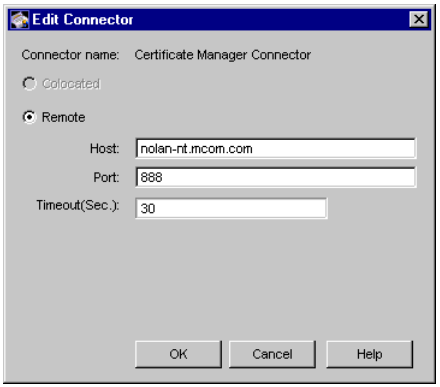


4. In the "List of connectors" select the connector:

   - If you are connecting the Registration Manager to a Certificate Manager, select Certificate Manager Connector and click Edit.
   - If you are connecting the Registration Manager to another Registration Manager, select Registration Manager Connector and click Edit.
   - If you are connecting the Registration Manager to a Data Recovery Manager, select Data Recovery Manager Connector and click Edit.

   For the purposes of completing these instructions, let us assume you selected Certificate Manager Connector.

The Edit Connector dialog box appears.



5. Select the Enable checkbox to enable the connector configuration.

6. Select Remote, and enter the appropriate information:

   Host. Type the full host name of the subsystem that trusts this Registration Manager; in this case, it would be the host name of the Certificate Manager. The Registration Manager uses this name to locate the Certificate Manager. The format for the host name must be as follows:

   `<machine_name>.<your_domain>.<domain>`

   Port. Type the number of the TCP/IP port at which the Certificate Manager will listen to requests from the trusted Registration Manager. The default port designated for communication between a trusted Registration Manager and a subsystem is the agent's port. See "Agent Port" on page 157.

   Timeout. Connection timeout. By default, it is 30 seconds.

   The sample screen above shows how to connect the Registration Manager to a Certificate Manager running on a host called `nolan-nt.mcom.com` listening for HTTPS requests on port `888`.

7. Click OK.

   You are returned to the Connectors tab.

8. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, stop and restart the server.

Setting Up a Certificate Manager as a Trusted Manager

You can set up a Certificate Manager to function as a trusted manager to a remote Data Recovery Manager. The setup process involves the following steps:

- Step 1. Find the Required Information
- Step 2. Create a User Entry for the Certificate Manager
- Step 3. Copy the Certificate Manager's Certificate to the Internal Database
- Step 4. Check the Certificate Database for the CA Certificate
- Step 5. Configure Certificate Manager's Connector Settings

Step 1. Find the Required Information

Before setting up a Certificate Manager to function as a trusted manager to a Data Recovery Manager:

- Note identifying information, such as the instance ID and host name of the Certificate Manager.
- Make sure that the Certificate Manager has the certificate you want it to use for SSL client authentication to the Data Recovery Manager that will trust it; by default, the Certificate Manager uses its SSL server certificate for this purpose. The certificate must be currently valid; the certificate must not have expired, been revoked, or been signed by an authority *untrusted* by the subsystem. For details, see "Trusted Manager's Certificate for SSL Client Authentication" on page 184.
- Locate the certificate in base-64 encoded format. Copy the certificate, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file.
- Identify the Data Recovery Manager to which you want to connect the Certificate Manager. Note details, such as the host name and port number of that Data Recovery Manager.

Step 2. Create a User Entry for the Certificate Manager

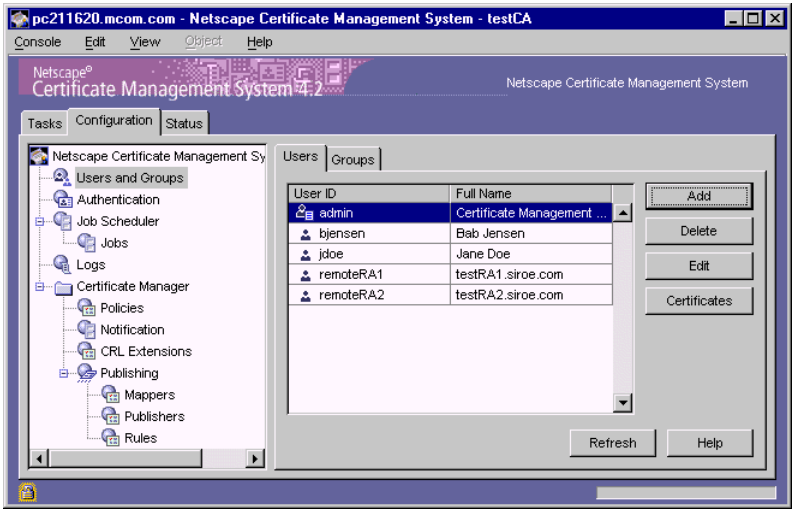
In this step, you create a privileged-user entry for the Certificate Manager in the internal database of the Data Recovery Manager. As a part of creating this entry, you also add the user entry to the `Trusted Managers` group in order to give the entry access privileges to the agent port of the Data Recovery Manager.

To create a user entry with appropriate access privileges for a Certificate Manager:

1. Log in to the CMS window for the Data Recovery Manager (see "Logging In to the CMS Window" on page 78).

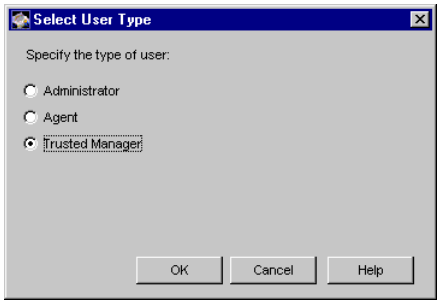
2. In the navigation tree, select Users and Groups.

   The Users tab appears in the right pane.



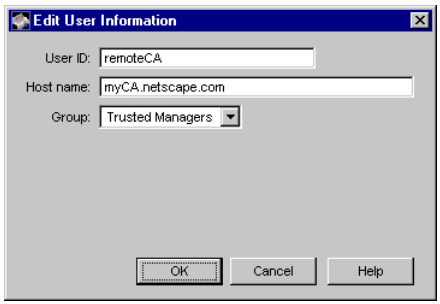
3. Click Add.

The Select User Type window appears.



4. Select Trusted Manager and click OK.

   The Edit User Information window appears.



5. Specify information as appropriate.

   The information you enter here is to help you keep track of the Certificate Manager; the Data Recovery Manager never uses it. The Data Recovery Manager relies solely on the Certificate Manager's SSL server certificate (which you will add in Step 3) for authentication.

   User ID. Type the Certificate Manager's instance ID (or any other ID that will help you identify the Certificate Manager in the list of privileged users). The ID can be an alphanumeric string of up to 255 characters.

   Host name. Type the fully qualified host name of the Certificate Manager. The host name can be an alphanumeric string of up to 255 characters. It must be in this form: `<machine_name>.<your_domain>.<domain>`

   Group. Select Trusted Managers; for more information about this group, see "Group for Trusted Managers" on page 189.

6. Click OK.

   You are returned to the Users tab. The Certificate Manager you just added is displayed in the list of users.

What you do next depends on whether you have the Certificate Manager's SSL server certificate:

- If you copied the Certificate Manager's certificate in base-64 encoded form to a text file, proceed to Step 3. (For details on getting this certificate, see "Trusted Manager's Certificate for SSL Client Authentication" on page 184.)
- Otherwise, skip to Step 5. You can add the certificate later, following the instructions in "Changing a Privileged User's Certificate" on page 220.

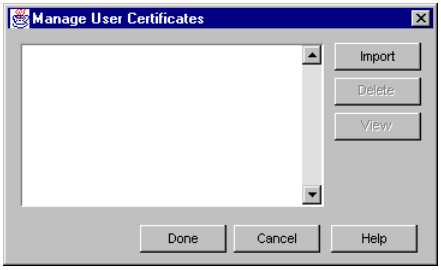
Step 3. Copy the Certificate Manager's Certificate to the Internal Database

In this step, you add the Certificate Manager's SSL server certificate to the internal database of the Data Recovery Manager and associate the certificate with the user entry you created in Step 2.

To store the Certificate Manager's SSL server certificate in the internal database of the subsystem:

1. In the Users tab, select the user entry you just added for the Certificate Manager and click Certificates.

   The Manage User Certificates window appears.

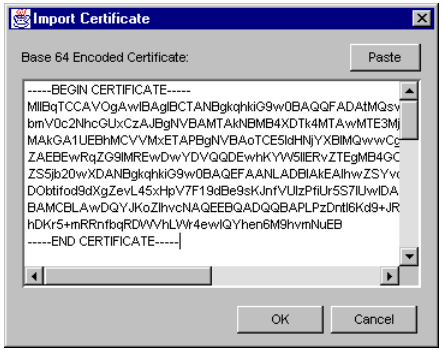


2. Click Import.

   The Import Certificate window appears.

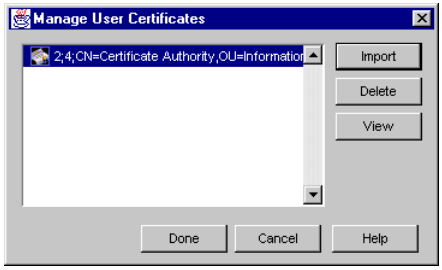
3. Click inside the text area, and paste the Certificate Manager's certificate in base-64 encoded form.

Be sure to include the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines.



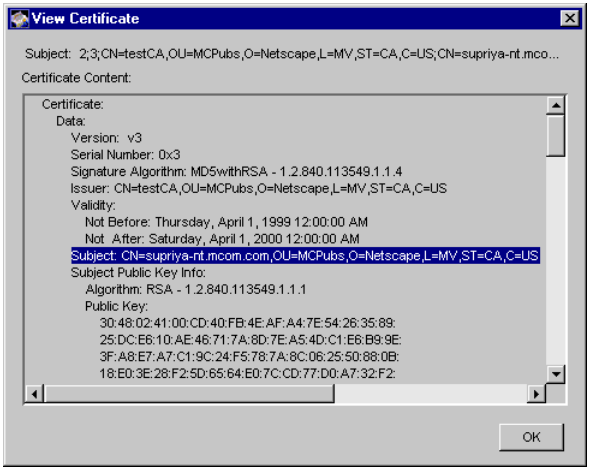
4. Click OK.

   You are returned to the Manage User Certificates window. The certificate you imported should now be listed in this window.



5. To view the certificate you imported, select it and click View.

The certificate information appears. Verify that the certificate you added is the correct one.



6. Click Done.

   You are returned to the Users tab.

Step 4. Check the Certificate Database for the CA Certificate

The issuer of the Certificate Manager's certificate that you added in Step 3 must be *trusted* by the Data Recovery Manager that services the key archival requests initiated by the Certificate Manager. Make sure that this CA's certificate exists in the Data Recovery Manager's certificate database (internal) and that it is trusted. To check whether the CA's certificate exists in the Data Recovery Manager's certificate database, follow the instructions in "Viewing the Certificate Database Content" on page 295.

- If the CA certificate isn't listed, follow the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260 and add the certificate to the certificate database.
- If the CA's certificate is listed but *untrusted*, follow the instructions in "Changing the Trust Settings of a CA Certificate" on page 299 and change the trust setting to *trusted*.

Step 5. Configure Certificate Manager's Connector Settings

In this step you configure the connector settings of the Certificate Manager. This enables the Certificate Manager to utilize the proprietary HTTPS connectors to communicate with the Data Recovery Manager (following successful SSL client authentication).

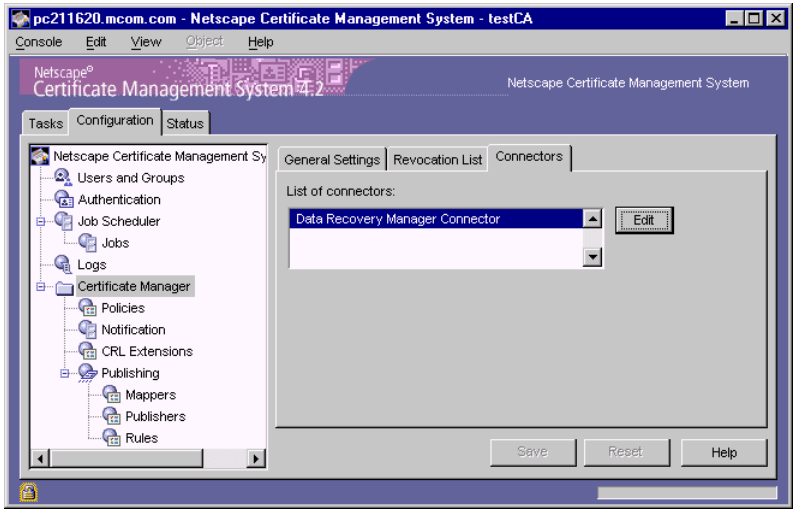
Note that during the installation of a Data Recovery Manager, you were prompted to specify the host name and port number of the Certificate Manager to which the Data Recovery Manager will be connected. If you specified this information, you are not required to go through this step. However, it is recommended that you verify the connector setting and make sure that the information you entered during installation is correct.

1. Log in to the CMS window for the Certificate Manager (see "Logging In to the CMS Window" on page 78).

2. In the navigation tree, select Certificate Manager.

   The General Settings tab appears in the right pane.

3. Select the Connectors tab.



4. In the "List of connectors" select Data Recovery Manager Connector and click Edit.

The Edit Connector dialog box appears.

5. Select the Enable checkbox to the enable the connector configuration.

6. Select Remote, and enter the appropriate information:

   Host. Type the full host name of the Data Recovery Manager that trusts this Certificate Manager. The Certificate Manager uses this name to locate the Data Recovery Manager. The format for the host name must be in the `<machine_name>.<your_domain>.<domain>` form.

   Port. Type the number of the TCP/IP port at which the Data Recovery Manager will listen to requests from the trusted Certificate Manager. The port designated for communication between a trusted Certificate Manager and a Data Recovery Manager is the agent port. See “Agent Port” on page 157.

   Timeout. Connection timeout. By default, it is 30 seconds.

   The sample screen above shows how to connect the Certificate Manager to a Data Recovery Manager running on a host called `testKRA.siroe.com` listening for HTTPS requests at port `8000`.

7. Click OK.

   You are returned to the Connectors tab.

8. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, stop and restart the server.

Changing Privileged-User Information

You can change privileged-user information in several ways:

- To change the login information of a privileged user, see “Changing a Privileged User’s Login Information” on page 219.
- To add or remove certificates of a privileged user, see “Changing a Privileged User’s Certificate” on page 220.
- To change the group membership or access permissions of a privileged user, see “Changing Members in a Group” on page 221.

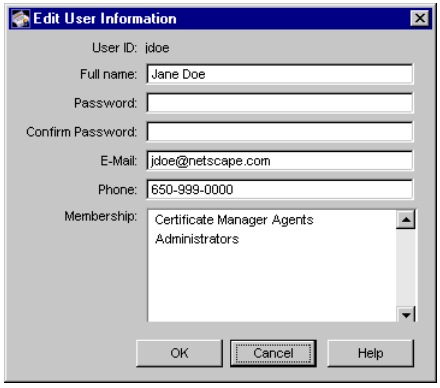
Changing a Privileged User’s Login Information

To change a privileged user’s login information:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).
2. In the navigation tree, select Users and Groups.

   The Users tab appears in the right pane.
3. In the User ID list, select the user you want to edit, and click Edit.

   The Edit User Information window appears.



4. Make the appropriate modifications.

   If you need details about individual fields, see “Setting Up Privileged Users” on page 190.

5. Click OK.

   You are returned to the Users tab.

6. Click Refresh to view the updated configuration.

Changing a Privileged User’s Certificate

To change a privileged user’s certificate:

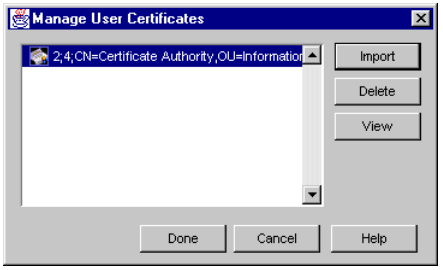
1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. In the navigation tree, select Users and Groups.

   The Users tab appears in the right pane.

3. In the User ID list, select the user whose certificate information you want to change, and click Certificates.

   The Manage User Certificate window appears.



4. Take the appropriate action:

   - To view a certificate, select the certificate and click View.
   - To delete a certificate, select the certificate and click Delete.

- To add a new certificate for this user to the internal database, click Import. In the Import Certificate window that appears, paste the new certificate in the text area. Be sure to paste the entire base-64 encoded block, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines. For details on getting the user's certificate, see "Agent's Certificate for SSL Client Authentication" on page 175.

5. Click Done.

   You are returned to the Users tab.

6. Click Refresh to view the updated configuration.

Changing Members in a Group

You can add or remove members from all groups. Keep in mind that the group for administrators must have at least one user entry. For details, see "Groups and Their Privileges" on page 186.

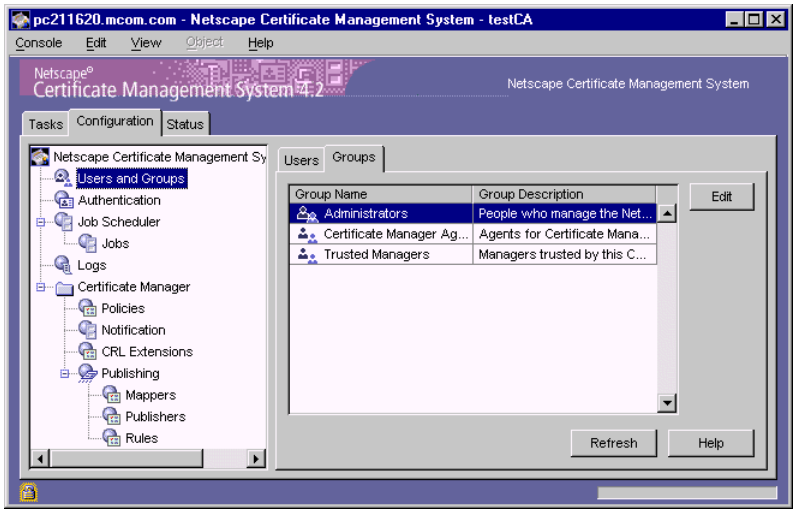
To change a group's members:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

2. In the navigation tree, select Users and Groups.

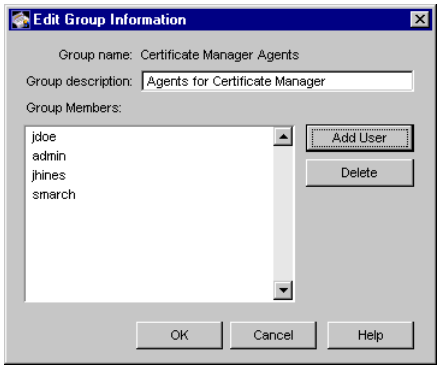
   The Users tab appears in the right pane.

3. Click the Groups tab.



4. In the Group Name list, select the group you want to change, and click Edit.

   The Edit Group Information window appears.



5. Make the appropriate changes:

   - To change the group description, type a new description in the “Group description” field.
   - To remove a user from the group, select the user and click Delete.

- To add users, click Add User. In the User Selection window that appears, select the users you want to add and click OK. You are returned to the Edit Group Information window.

6. Click OK when you are done with the changes.

   You are returned to the Groups tab.

7. Click Refresh to view the updated configuration.

Deleting a Privileged User

You can delete privileged users from the internal database. Deleting a user from the internal database deletes that user from all groups to which the user belongs. If you want to delete a user from specific groups, you should modify the appropriate groups; for details, see "Changing Members in a Group" on page 221.

To delete a privileged user from the internal database:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

2. In the navigation tree, select Users and Groups.

   The Users tab appears in the right pane.

3. In the User ID list, select the user you want to delete, and click Delete.

4. When prompted, confirm your action.

   If you click OK, the user entry is deleted from the internal database.

5. Click Refresh to view the updated configuration.

Chapter

8

Keys and Certificates

The main subsystems of Netscape Certificate Management System (CMS)—the Certificate Manager, Registration Manager, and Data Recovery Manager—use certificates for authentication during SSL-enabled communication. For example, when a Registration Manager forwards a certificate issuance request to a Certificate Manager for signing, the Certificate Manager expects the Registration Manager to have performed SSL client authentication before processing the request.

When you installed Certificate Management System, the installation program prompted you to generate the required certificates for the subsystems you chose to install. This chapter provides an overview of those certificates and it explains how to perform operations such as renewing the existing certificates before their validity period expires, getting new certificates for the subsystems, adding trusted CA certificates and certificate chains to the CMS trust database, and changing the trust setting of CA certificates. The chapter also explains the certificate Setup Wizard, which automates the process of requesting and installing certificates.

The chapter has the following sections:

Keys and Certificates for the Main Subsystems

This section explains the various certificates required and used by the Certificate Manager, Registration Manager, and Data Recovery Manager. The key pairs that correspond to certificates used by these subsystems can be stored either in an internal or an external token, or in both. It depends on the token you chose for the generation and storage of the keys and certificates. For information on tokens, see "Tokens for Storing Keys and Certificates" on page 235.

Certificate Manager's Key Pairs and Certificates

The Certificate Manager uses the following key pairs and corresponding certificates:

- CA Signing Key Pair and Certificate
- SSL Server Key Pair and Certificate

CA Signing Key Pair and Certificate

Every Certificate Manager you installed has a certificate, identified as the *Certificate Manager CA signing certificate*, whose public key corresponds to the private key the Certificate Manager uses to sign the certificates it issues. The first time you generated this certificate is when you installed the Certificate Manager. The default nickname for the certificate is

`caSigningCert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Certificate Manager is installed, and the default validity period for the certificate is two years.

The subject name of the CA signing certificate reflects the name of your certificate authority (CA) as specified during the installation. All certificates signed or issued by the Certificate Manager include this name to identify the issuer of the certificate.

Important You cannot change the CA name; doing so would make all previously issued certificates invalid.

The Certificate Manager's status as a root or subordinate CA is determined by whether its CA signing certificate is self-signed or is signed by another CA.

- If the Certificate Manager is a root CA, its CA signing certificate is self-signed.
- If the Certificate Manager is a subordinate CA, its CA signing certificate is signed by another CA, usually the one that is a level above in the CA hierarchy (which may or may not be a root CA). If you have deployed the Certificate Manager as a subordinate CA in a CA hierarchy, you must import your root CA's signing certificate into individual clients and servers before you can use the Certificate Manager to issue certificates to them.

As an administrator, you must make sure that the private key that corresponds to the CA signing certificate is adequately protected. This includes protecting it from damage (in other words, by archiving and backing up the key) as well as protecting it from unauthorized access or use. The password that protects the token containing this key must be carefully guarded. Access to the token itself should be limited. (See "Tokens for Storing Keys and Certificates" on page 235.)

- If the key is in the internal token (the `key3.db` file), make sure that only you or authorized administrators have access to this file. It's also important to know if the file is stored on backup tapes or is otherwise available for someone to intercept. Because the destruction of a private key in a disk crash can be disastrous if you are depending upon that key for a hierarchy of certificate authorities, backing up your key data is commensurately important. If you do make copies of your keys, however, you must protect your backups with the same level of security that you use for protecting your original keys.
- If the key is in an external token, such as a smart card, keep it in a locked facility.

Also, periodically change the password that protects this key. See "Changing a Token's Password" on page 240.

Like any other certificate, the Certificate Manager's CA signing certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

The CA signing key pair must be well protected to ensure that it is never compromised. However, if you know or suspect that the key pair has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

Important Reissuing the Certificate Manager's CA signing certificate with a new key pair invalidates all certificates that have been signed by the old key pair.

SSL Server Key Pair and Certificate

Every Certificate Manager you have installed has at least one *SSL server certificate*. The first time you generated this certificate is when you installed the Certificate Manager. The default nickname for the certificate is `Server-Cert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Certificate Manager is installed.

The Certificate Manager's SSL server certificate was issued by the CA to which you submitted the certificate signing request. You might have submitted the request to the Certificate Manager itself, another internally deployed CA, or a public CA. To find out the issuer name, follow the instructions in "Viewing the Certificate Database Content" on page 295.

The Certificate Manager uses its SSL server certificate to do SSL server-side authentication to the following:

- Netscape Console
- The End-Entity Services interface (the HTTPS port)
- The Certificate Manager Agent Services interface

By default, the Certificate Manager uses a single SSL server certificate for authentication purposes. However, you can request and install additional SSL server certificates for the Certificate Manager. For example, you can configure the Certificate Manager to use separate server certificates for authenticating to

Netscape Console, the End-Entity Services interface, and the Certificate Manager Agent Services interface. For instructions, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269.

If you configure the Certificate Manager for SSL-enabled communication with the publishing directory, it also uses its SSL server certificate for SSL client authentication to the publishing directory; this is the default configuration. You can configure the Certificate Manager to use an alternate certificate for this purpose; see "Getting an SSL Client Certificate for a Subsystem" on page 271.

If you configure the Certificate Manager to function as a *trusted manager* to a Data Recovery Manager, the Certificate Manager also uses its SSL server certificate for SSL client authentication to the Data Recovery Manager. For details on trusted managers, see "Trusted Managers" on page 181. You can also configure the Certificate Manager to use an alternate certificate for this purpose; see "Getting an SSL Client Certificate for a Subsystem" on page 271.

Like any certificate, the SSL server certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

The SSL server key pair must be well protected to ensure that it is never compromised. However, if you know or suspect that the key pair has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

Note If you have installed the Certificate Manager with a Data Recovery Manager, both subsystems use the same SSL server certificate.

Registration Manager's Key Pairs and Certificates

The Registration Manager uses the following certificates:

- Signing Key Pair and Certificate
- SSL Server Key Pair and Certificate

Signing Key Pair and Certificate

Every Registration Manager you have installed has a certificate, identified as the *Registration Manager signing certificate*, whose public key corresponds to the private key the Registration Manager uses to sign certificate requests before sending them to the Certificate Manager for signing. The Registration Manager's signature provides persistent proof to the Certificate Manager that the Registration Manager has processed the request. The first time you generated this certificate is when you installed the Registration Manager. The default nickname for the certificate is `raSigningCert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Registration Manager is installed.

The Registration Manager's signing certificate was issued by the CA to which you submitted the certificate signing request. You might have submitted the request to an internally deployed CA or a public CA. To find out the issuer name, follow the instructions in "Viewing the Certificate Database Content" on page 295.

If you configure the Registration Manager to function as a *trusted manager* to another subsystem, the Registration Manager uses its signing certificate for SSL client authentication to the subsystem; this is the default configuration. For details, see "Trusted Manager's Certificate for SSL Client Authentication" on page 184.

Like any certificate, the signing certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

The Registration Manager's signing key pair must be well protected to ensure that it is never compromised. However, if you know or suspect that the key pair has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

SSL Server Key Pair and Certificate

Every Registration Manager you have installed has at least one *SSL server certificate*. The first time you generated this certificate is when you installed the Registration Manager. The default nickname for the certificate is `Server-Cert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Registration Manager is installed.

The Registration Manager's SSL server certificate was issued by the CA to which you submitted the certificate signing request. You might have submitted the request to an internally deployed CA or a public CA. To find out the issuer name, follow the instructions in "Viewing the Certificate Database Content" on page 295.

The Registration Manager uses its SSL server certificate to do SSL server-side authentication to the following:

- Netscape Console
- The end entity services interface (the HTTPS port)
- The Registration Manager Agent Services interface

By default, the Registration Manager uses a single SSL server certificate for authentication purposes. However, you can request and install additional SSL server certificates for the Registration Manager. For example, you can configure the Registration Manager to use separate server certificates for authenticating to Netscape Console, the end entity services interface, and the Registration Manager Agent Services interface. For instructions, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269.

Like any certificate, the SSL server certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

The SSL server key pair must be well protected to ensure that it is never compromised. However, if you know or suspect that the key pair has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

Note If you installed the Registration Manager with a Data Recovery Manager, both subsystems use the same SSL server certificate.

Data Recovery Manager's Key Pairs and Certificates

The Data Recovery Manager uses the following certificates:

- Transport Key Pair and Certificate
- Storage Key Pair
- SSL Server Key Pair and Certificate

Transport Key Pair and Certificate

Every Data Recovery Manager you have installed has a *Data Recovery Manager transport certificate*. The public key of the key pair that is used to generate the transport certificate is used by the client software to encrypt an end user's *encryption private key* before it is sent to the Data Recovery Manager for archival; only those clients capable of generating dual-key pairs (one for signing and one for encryption) use the transport certificate. For more information on how this certificate is used, see "Key Archival Process" on page 1118.

The first time you generated this certificate is when you installed the Data Recovery Manager. The default nickname for the certificate is `kraTransportCert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Data Recovery Manager is installed.

The transport certificate was issued by the CA to which you submitted the certificate signing request. You might have submitted the request to the Certificate Manager that is installed in the same instance, internally deployed another CA, or a public CA. To find out the issuer name, follow the instructions in "Viewing the Certificate Database Content" on page 295.

Like any certificate, the transport certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

The transport key pair must be well protected to ensure that it is never compromised. However, if you know or suspect that the key pair has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

Storage Key Pair

Every Data Recovery Manager you have installed has a *Data Recovery Manager storage key pair*. The first time you generated this key pair is when you installed the Data Recovery Manager.

The Data Recovery Manager uses the public component of this key pair to encrypt (or wrap) end users' encryption private keys during the key archival operation; it uses the private component to decrypt (or unwrap) the archived key during the recovery operation. That is, the public key is used to encrypt the key repository the server uses to store end users' encryption private keys. For more information on how this key pair is used, see "Recovering Encrypted Data" on page 1115.

Note that the public component of the storage key pair is not certified; there is no certificate that corresponds to the public key.

Keys encrypted with the storage key can be retrieved only by authorized key recovery agents. For details, see "Key Recovery Agents and Their Passwords" on page 1122.

SSL Server Key Pair and Certificate

Every Data Recovery Manager you have installed has at least one *SSL server certificate*. The first time you generated this certificate is when you installed the Data Recovery Manager. The default nickname for the certificate is `Server-Cert cert-<instance_id>`, where `<instance_id>` identifies the CMS instance in which the Data Recovery Manager is installed.

The Data Recovery Manager's SSL server certificate was issued by the CA to which you submitted the certificate signing request. You might have submitted the request to the Certificate Manager that is installed in the same instance, an internally deployed CA, or a public CA. To find out the issuer name, follow the instructions in "Viewing the Certificate Database Content" on page 295.

The Data Recovery Manager uses its SSL server certificate to do SSL server-side authentication to the following:

- Netscape Console
- The end entity services interface (the HTTPS port)
- The Data Recovery Manager Agent Services interface

By default, the Data Recovery Manager uses a single SSL server certificate for authentication purposes. However, you can request and install additional SSL server certificates for the Data Recovery Manager. For example, you can configure the Data Recovery Manager to use separate server certificates for authenticating to Netscape Console, the end entity services interface, and the Data Recovery Manager Agent Services interface. For instructions, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269.

Like any certificate, the SSL server certificate has a validity period. You must renew the certificate before it expires. For instructions on renewing the certificate, see "Renewing Certificates for the Subsystems" on page 286.

If you know or suspect that the key pair for the SSL server certificate has been compromised, reissue the certificate with a new key pair. For instructions on getting a new certificate, see "Getting New Certificates for the Subsystems" on page 277.

Note If you installed the Data Recovery Manager with a Certificate Manager or Registration Manager, both subsystems use the same SSL server certificate.

Tokens for Storing Keys and Certificates

A token is a hardware or software device that performs cryptographic functions and optionally stores public-key certificates, cryptographic keys, and data defined by the application using the cryptographic services. Alternatively, a token can also be considered as a device that you can use to generate and store your key pairs and corresponding certificates.

Certificate Management System defines two types of tokens, *internal* and *external*, for storing key pairs and certificates that belong to the Certificate Manager, Registration Manager, and Data Recovery Manager.

Note Only those who have the password that protects a token can access it. For information on changing the password that protects a token, use the command-line utility called the *Key Database Tool*; for details about this utility, see Appendix E, "Key Database Tool" in the online version of this document. To locate the document, see "Where to Go for Related Information" on page 39.

Internal Token

An internal (software) token refers to a pair of software files, usually called *certificate database* and *key database*, that Certificate Management System uses to generate and store its key pairs and certificates. Certificate Management System automatically generates these files in the file system of its host machine when you choose to use the internal token for the first time. These files were created for you during CMS installation if you chose to use the internal token for key-pair generation.

In the CMS host system, the certificate database is identified by the name `cert7.db`; the key database is identified by the name `key3.db`. You can find both these files at this location:

```
<server_root>/cert-<instance_id>/config
```

External Token

An external (hardware) token refers to an external hardware device, such as a smart card, FORTEZZA card, or other crypto card, that Certificate Management System uses to generate and store its key pairs and certificates. Certificate Management System supports any hardware tokens that are compliant with PKCS #11 version 2.01. For details, see the information provided at this URL:

```
http://developer.netscape.com/support/faqs/pkcs_11.html
```

If you haven't already done so, consider using external tokens for generating and storing the key pairs and certificates used by Certificate Management System. These devices represent another security measure you can take to safeguard private keys because hardware tokens are sometimes considered more secure than software tokens. For additional details, check the literature provided by hardware-token vendors.

Installing External Tokens

To use external encryption devices or tokens, you need to take the following steps:

- Step 1. Install the Cryptographic Device
- Step 2. Install the PKCS #11 Module

Step 1. Install the Cryptographic Device

To install the drivers provided by the device manufacturer, follow the instructions that came with the device. When you install a hardware token, you are given an opportunity to name it; be sure to use a name that will help you identify the token later.

Step 2. Install the PKCS #11 Module

PKCS #11 is a standard set of APIs and shared libraries used by Netscape and a number of encryption vendors. PKCS #11 isolates an application from the details of the cryptographic device, thus enabling the application to provide a unified interface for PKCS #11-compliant cryptographic devices.

The PKCS #11 module implemented in Certificate Management System (in Netscape Administration Server) enables it to support cryptographic devices supplied by many different manufacturers. Specifically, it allows Certificate Management System to plug in shared libraries or DLLs supplied by manufacturers of external encryption devices and use them for generating and storing keys and certificates for the Certificate Manager, Registration Manager, and Data Recovery Manager.

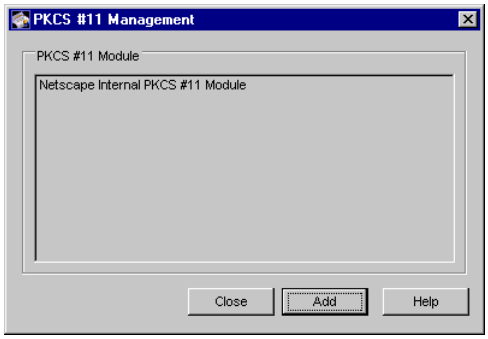
There are two ways in which you can install a PKCS #11 module, by using the interface provided within Netscape Console or by using the command-line utility named `modutil`. Both the methods are documented in the sections that follow.

- To install the PKCS #11 module using Netscape Console:

  1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

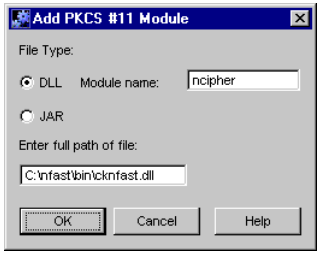
  2. From the Console menu, choose Manage PKCS#11.

     The PKCS #11 Management window appears.



  3. Click Add.

     The Add PKCS #11 Module window appears.



  4. Enter information as appropriate. If you choose JAR as your file type, you are required to provide the path to the JAR file that contains the DLLs. If you choose DLL as your file type, in addition to the path to the DLL you are also required to provide a name for the module you're attempting to install (so as to help you identify it easily later). The sample in the figure shows how you would install an nCipher™ token.

  5. Click OK.

- To install the PKCS #11 module using the `modutil` tool (which is located at `<server_root>/shared/bin`):

  1. Locate the CMS instance for which you want to install the PKCS #11 module.

  2. Open a terminal window.

  3. Go to the configuration directory of Administration Server; it is located here: `<server_root>/admin-serv/config`

  4. At the prompt, enter this command:

     ```
     <server-root>/shared/bin/modutil -dbdir . -nocertdb
     -create
     ```

     This creates the required security module database file (`secmod.db`) in the configuration directory.

  5. At the prompt, enter this command:

     ```
     <server_root>/shared/bin/modutil -dbdir . -nocertdb
     -add <module_name> -libfile <library_file>
     ```

     `<library_file>` specifies the path to the DLL or other library file containing the implementation of the PKCS #11 interface module.

     `<module_name>` specifies the name of the PKCS #11 module (which you specified in Step 1 when you installed the drivers).

     For example, if you are installing a Litronic™ token, the command would look like this:

     ```
     <server_root>/shared/bin/modutil -dbdir . -nocertdb
     -add CryptOS -libfile core32
     ```

  6. Copy the new `secmod.db` (`secmodule.db` on Unix) to the following location: `<server-root>/cert-<instance_id>/config/`

     This overwrites the old `secmod.db` in this directory.

Managing Tokens Used by the Subsystems

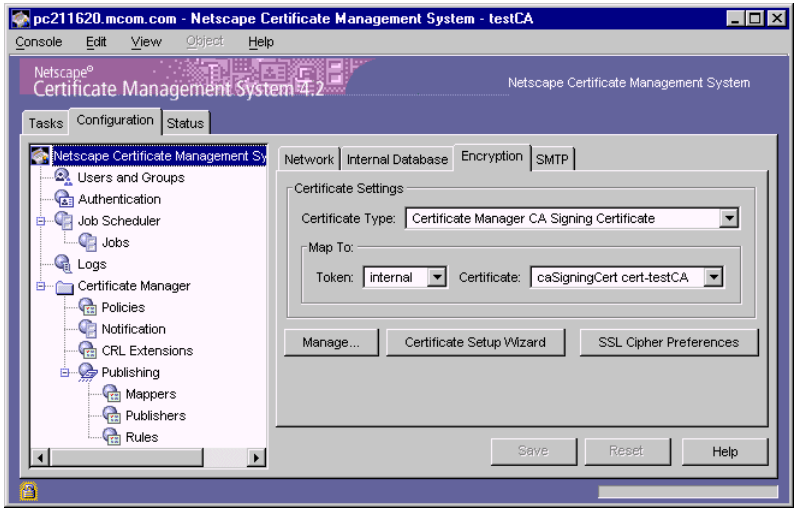
There are two main tasks involved in managing the tokens used by Certificate Management System:

- Viewing Tokens
- Changing a Token's Password

Viewing Tokens

To view a list of the tokens currently installed for a CMS instance:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab, and then in the right pane, select the Encryption tab.



3. In the Map To section, check the Token drop-down list.

It shows the names (as specified when the tokens were installed) of external tokens installed for the currently selected CMS instance. For information on installing external tokens, see "External Token" on page 236.

Changing a Token's Password

The token, internal or external, that stores the key pairs and certificates for the subsystems is protected (encrypted) by a password. To decrypt the key pairs or to gain access to them, you must enter that password. The first time you specified this password is when you used the token the first time, most likely during CMS installation.

It is good security practice to periodically change the password that protects your server's keys and certificates; changing the password periodically minimizes the risk of someone finding out the password. To change a token's password, use the command-line utility called the *Key Database Tool*; for details about this utility, see Appendix E, "Key Database Tool" in the online version of this document.

Note that the single sign-on password cache stores the passwords for tokens in order to start the server using a single password; for details, see "Required Start-up Information" on page 128. Whenever you change the password, the cache is updated with the new password.

Hardware Cryptographic Accelerators

Certificate Management System allows you to use hardware cryptographic accelerators with external tokens. Many of the accelerators provide the following security features:

- Fast SSL connections—speed is important if you want your Certificate Manager, Registration Manager, or Data Recovery Manager to be able to accommodate a high number of simultaneous enrollment or service requests.

- Hardware protection of private keys—these devices behave like smart cards, in that they do not allow the private keys to be copied or removed from the hardware token. This is important if you are concerned about the risks associated with key theft from an active attacker of your online Registration Manager or Certificate Manager.

For a list of vendors that supply hardware cryptographic accelerators, see this URL: `http://www.iplanet.com/products/infrastructure/dir_security/cert_sys/index.html`

Certificate Setup Wizard

Certificate Management System provides a wizard, called the *Certificate Setup Wizard*, which automates the process of requesting and installing the certificates required by Certificate Management System. (For a description of these certificates, see "Keys and Certificates for the Main Subsystems" on page 226.)

The Certificate Setup Wizard is integrated into the CMS window, enabling you to accomplish the following tasks:

- Renew certificates of the Certificate Manager, Registration Manager, or Data Recovery Manager installed in the currently selected CMS instance; renewing a certificate means getting a new certificate with the same subject name and public and private key material as that of the existing certificate, but with an extended validity period.
- Request and install new certificates for the Certificate Manager, Registration Manager, and Data Recovery Manager installed in the currently selected CMS instance; reissuing or requesting a new certificate means getting a certificate based on a new public and private key pair.
- Install CA certificates in the trust database of the currently selected CMS instance.
- Install CA certificate chains in the trust database of the currently selected CMS instance.

When you start the wizard, which you do by clicking the Certificate Setup Wizard button in the Encryption tab of the CMS window (see the figure on page 240), you are asked to specify whether you want to request or install a certificate. The wizard presents you with the screens appropriate to your choice and walks you through the entire process.

For installing certificates, except for cases when the certificate is self-signed by the CA, you will need to run the wizard twice: once, to request the certificate and once to install the certificate. The reason for this is, if you submit the certificate request to a non-local CA, you will have to wait for the certificate until it is delivered to you.

The following sections explain the process of requesting and installing a certificate by using the Certificate Setup Wizard:

- Using the Wizard to Request a Certificate
- Using the Wizard to Install a Certificate or Certificate Chain

For instructions on getting new certificates, see "Getting New Certificates for the Subsystems" on page 277. For instructions on renewing existing certificates, see "Renewing Certificates for the Subsystems" on page 286.

Using the Wizard to Request a Certificate

The Certificate Setup Wizard allows you to request any of the certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager installed in the currently selected CMS instance.

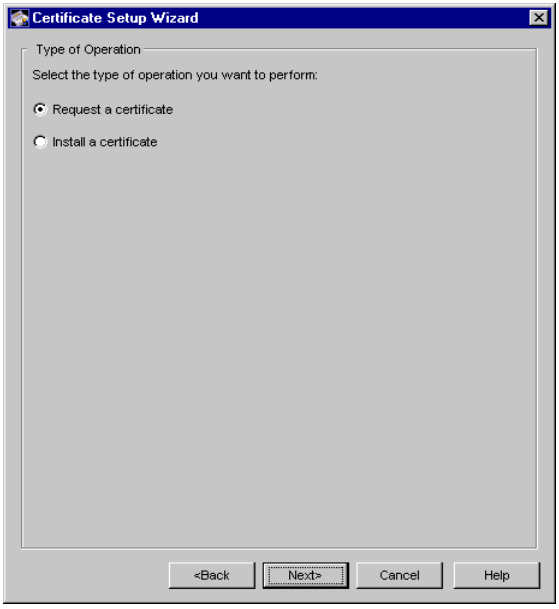
Using the wizard to request a certificate involves the following steps:

- Step 1. Select the Operation
- Step 2. Choose the Certificate
- Step 3. Specify the Key-Pair Information
- Step 4. Specify the Subject Name for the Certificate
- Step 5. Specify the Validity Period

- Step 6. Specify Extensions
- Step 7. Copy the Certificate Signing Request
- Step 9. Send the Certificate Signing Request to a CA
- Step 8. Check the Certificate Request Status

Step 1. Select the Operation

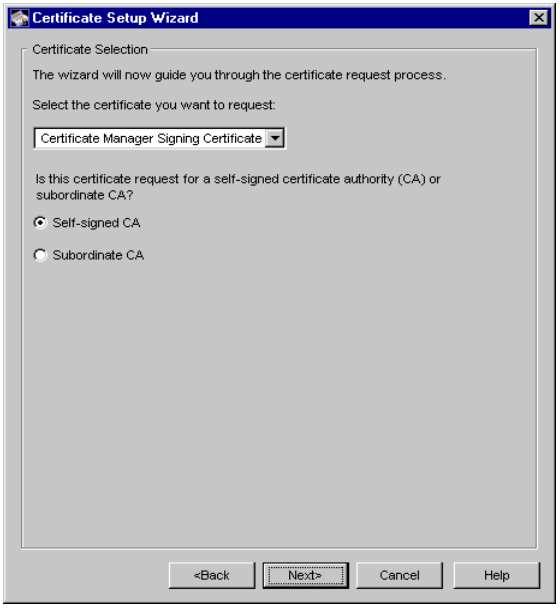
Indicate whether you want to request a certificate or install a certificate.



For the sake of completing the instructions that follow, assume that you chose to request a certificate.

Step 2. Choose the Certificate

Choose the certificate (by name) that you want to request.



The drop-down list shows various certificates used by the currently selected CMS instance. Choose the one you want to request. (If you need information on certificates used by Certificate Management System, see “Keys and Certificates for the Main Subsystems” on page 226.)

Which certificates you see in the list depends on the subsystems installed in the currently selected CMS instance. You may see a combination of the following options:

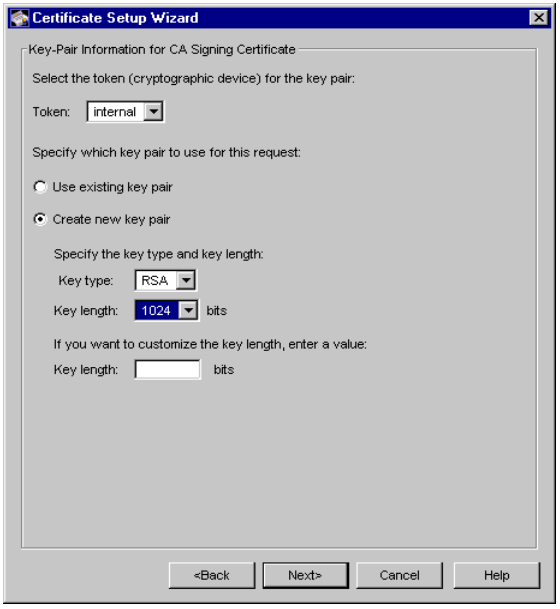
- If a Certificate Manager is installed, the list includes the Certificate Manager’s CA signing and SSL server certificates.
- If a Registration Manager is installed, the list includes the Registration Manager’s signing and SSL server certificates.
- If a Data Recovery Manager is installed, the list includes the Data Recovery Manager’s transport and SSL server certificate.

Depending on the certificate you want to generate, choose the one in the drop-down list:

- Certificate Manager Signing Certificate—choose this option if you want to request a signing certificate for the Certificate Manager installed in the currently selected CMS instance. If you choose this option, you must also specify whether the certificate request is for a self-signed CA (also known as the root CA) or a subordinate CA.
- Registration Manager Signing Certificate—choose this option if you want to request a signing certificate for the Registration Manager installed in the currently selected CMS instance.
- Data Recovery Manager Transport Certificate—choose this option if you want to request a transport certificate for the Data Recovery Manager installed in the currently selected CMS instance.
- SSL Server Certificate—choose this option if you want to generate an SSL server certificate request for the Certificate Manager, Registration Manager, or Data Recovery Manager installed in the currently selected CMS instance.

Step 3. Specify the Key-Pair Information

Specify the key-pair information for the certificate to be requested.



You need to identify the following:

- The token that contains the key pair for generating the certificate request

  The drop-down list shows the names of tokens currently installed for the selected CMS instance; these are the tokens you can use now.

  — The internal token is identified as *internal*. You should choose this option if the key pair for the certificate you chose in the previous step is stored in the local key database.

  — The names of external tokens vary, matching the names specified when the tokens were installed. You should choose this option if the key pair for the certificate you chose in the previous step is in an external cryptographic device. If you don't see the token you want to use, exit from the wizard, make sure the token is installed properly, restart the server, and repeat the process. For information on using or installing external tokens, see "Installing External Tokens" on page 236.

- The key pair for generating the certificate request

  You can choose to generate the certificate request based on an existing or a new key pair.

  — If you want to renew the certificate you selected in the previous step, use the existing key pair for generating the request. For example, you can extend the validity period of a certificate by renewing it.

  To generate a certificate request based on an existing key pair, select the token that contains the key pair you want to use for generating the request. The wizard automatically selects the key pair that corresponds to the certificate you chose in the previous step.

  — If you want a new certificate, use a new key pair for generating the request. For example, you may want to get a new SSL server certificate or may want to replace an existing certificate whose private key has been compromised.

  To generate a certificate request based on a new key pair, select the token that can generate the key pair you want to use for generating the request. For example, if you want to generate the key pair using an external cryptographic device, such as a smart card, select that as the token. In addition, you will be required to indicate details, such as the key algorithm and size for the key pair.

- The type and length of the key pair

  You are required to provide this information only if you chose to generate the certificate request based on a new key pair. For key type, you can choose RSA or DSA. Be sure to select a key type that the CA (to which you will later submit the request for signing) can certify.

  For key length, enter the size in bits.

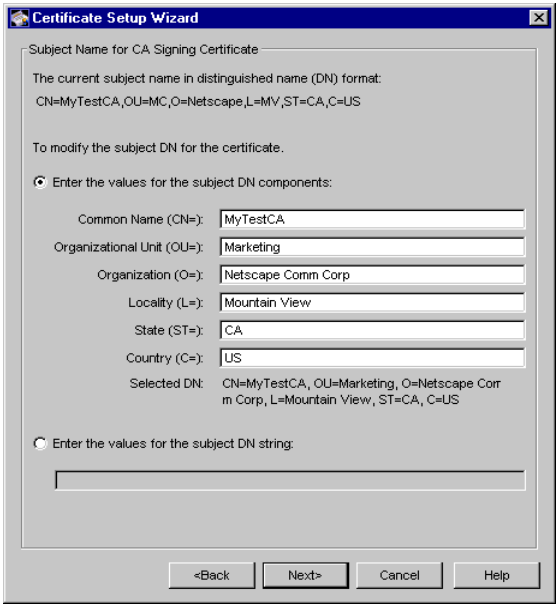
  — If the key type is RSA, the key size can be 512, 1024, 2048, 4098, or custom.

  — If the key type is DSA, the key size can be 512, 1024, or custom (must be in increments of 64 bit).

  Keep in mind that generating a new key pair takes time—the longer the key length the longer the time the wizard takes to generate it.

Step 4. Specify the Subject Name for the Certificate

Specify the subject name, in distinguished name (DN) format, for the certificate to be requested. Note that you will see this screen only if you chose to generate the certificate for a new key pair.



You can either enter values for individual DN attributes required to build the subject DN or build the complete subject DN string yourself. If you enter values for individual DN attributes, the wizard constructs the subject DN string.

If you want to enter values for individual DN components, provide the following information:

- Common name—enter the name as appropriate. The common name format, except for the SSL server certificate, can be a descriptive name of up to 255 characters; for example, you can name the Certificate Manager's signing certificate as "Root CA for ABC Corporation"; similarly, you can name the Registration Manager's signing certificate as "Registration Authority for North America". For a SSL server certificate, the name must be the fully qualified host name of Certificate Management System in this form:

  `<machine_name>.<your_domain>.<domain>`

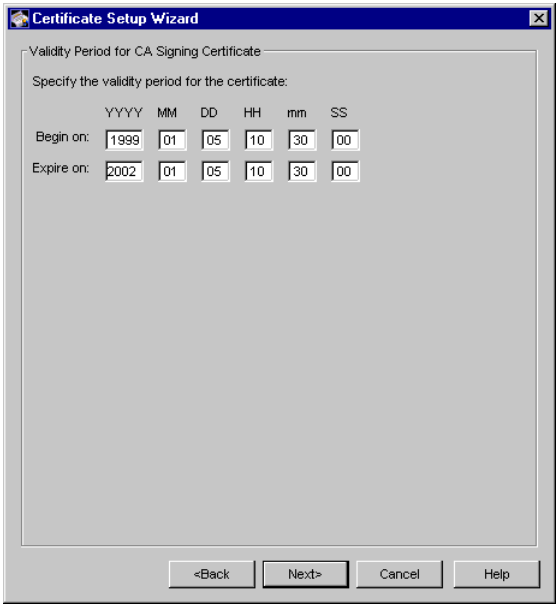
To determine the machine and domain names, go to Netscape Console, and locate the CMS host in the navigation tree.

- Organizational unit—enter the organizational unit you belong to.
- Organization—enter a description that identifies your organization.
- Locality—enter the name of the city where you business is located.
- State or province—enter the name of the state or province where your business is located.
- Country—enter the name of the country where your business is located.

Important When choosing subject names for certificates, be sure to follow the information provided in “Role of Distinguished Names in Certificates” on page 1166.

Step 5. Specify the Validity Period

You need to complete this step only if you chose to generate a self-signed CA certificate request.

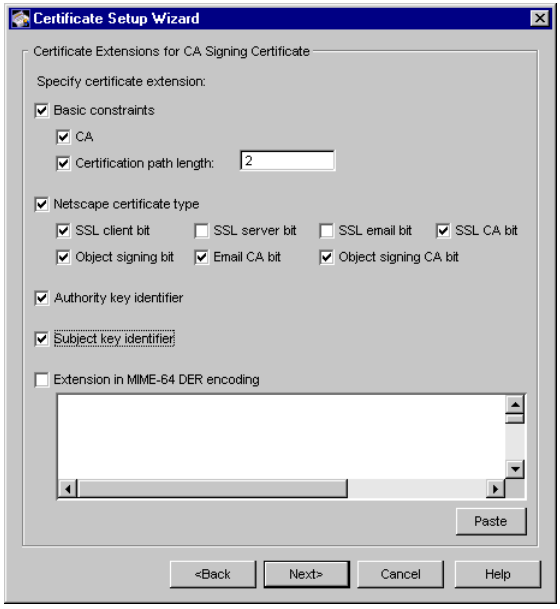


Specify the starting and ending dates of the validity period for the certificate request you want to generate. You can also specify the time at which the validity period should start and end on those dates. The date and time are in the form YYYYMMDD (Year, Month, Day) and HHmmSS (Hour, Minute, Second), in that order.

The default validity period is two years.

Step 6. Specify Extensions

You need to complete this step only if you chose to generate a CA signing certificate request for a Certificate Manager (deployed as either the root CA or a subordinate CA).



This screen allows you to set the standard X.509 version 3 extensions and Netscape-defined extensions for the certificate to be requested. The required extensions are chosen by default. If you want to change the default choices, be sure to read the general guidelines explained in "Certificate Extensions" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Also note that certificate extensions are required if you are setting up a hierarchy of certificate authorities (CAs). Subordinate CAs must have certificates that include the extension identifying them as either a subordinate SSL CA (which allows them to issue certificates for SSL) or a subordinate email CA (which allows them to issue certificates for secure email). If you disable certificate extensions, you will not be able to set up CA hierarchies. For more information on CA hierarchies, see "Certificate Hierarchies" in Appendix D of *Managing Servers with Netscape Console*.

You can set the following extensions:

- Basic constraints—select this option if you want to set any of the basic constraints extension bits in the certificate you are requesting. When you select the option, the associated fields are enabled. You should select the ones you want to set.
- Netscape certificate type—select this option if you want to set any of the Netscape Certificate Type extension bits in the certificate you are requesting. When you select the option, the associated fields are enabled. You should select the ones you want to set.
- Authority key identifier—select this option if you want to set the authority key identifier extension in the certificate you are requesting.
- Subject key identifier—select this option if you want to set the subject key identifier extension in the certificate you are requesting.
- Key usage—select this option if you want to set the key usage extension in the certificate you are requesting. If you choose this option, the digital signature (bit 0), non repudiation (bit 1), key Certificate Sign (bit 5), and CRL sign (bit 6) bits are set by default. The extension is marked critical as recommended by the PKIX standard and RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt` for a description of the Key Usage extension).
- Extension in MIME 64 DER encoding—select this option if you want to specify any custom extension. When you select the option, the associated text field is enabled. You should paste your extension (in MIME 64 DER encoded format) into the text field.

Certificate Management System provides tools that generate MIME-64 encoded blobs for many standard extensions. You can use these tools for generating MIME-64 encoded blobs for any extensions that you may want to include in CA and other certificate requests. For details about these tools, check the samples package at this location:

`<server_root>/cms_sdk/samples/exttools`

Note that the text field provided for pasting the extension in general accepts a single extension blob. If you want to add multiple extensions, you should use the `ExtJoiner` program, which is also provided as a sample in the CMS samples package at the above mentioned location.

Joining Multiple Custom Extensions Using ExtJoiner

The `ExtJoiner` is a program that joins a sequence of extensions together so that the final output can be used in the wizard text field for specifying multiple extensions; note that the program doesn't generate an extension, it only joins them. The command syntax for the `ExtJoiner` is as follows:

`java ExtJoiner <ext_file0> <ext_file1> ... <ext_fileN>`
where `<ext_file>` specifies the path, including the filename, to files that contain the base-64 encoded DER encoding of an X.509 extension.

The steps below outline how you can use the `ExtJoiner` to join multiple custom extensions:

Step 1. Write the appropriate Java programs for the extensions.

Step 2. Join the extensions using ExtJoiner. To do this:

1. Note the file paths to the files that contain the programs for extensions.
2. Open a command window.
3. Run the ExtJoiner, substituting the appropriate file paths. For example, if you have two extension files named `myExt1` and `myExt2` and have copied them to the same directory as the ExtJoiner, the command would look like this: `java ExtJoiner myExt1 myExt2`

   You should see a base-64 encoded blob, similar to the one below, of the joined extensions on screen:

   ```
   MEwwLgYDVR0lAQHBCQwIgYFKoNFBAMGClGC5EKDM5PeXzUGBi2CVyLNCQYFUTiB

   akowGgYDVR0SBBMwEaQPMA0xCzAJBgNVBAYTAlVT
   ```

4. Copy the encoded blob, without any modifications, to a file.

Step 3. Verify that the extensions are joined correctly before adding them to a certificate request. To do this, first you'll need to convert the binary data to ASCII format using the `AtoB` utility and then verify the binary data by dumping the contents of the base-64 encoded blob using the `dumpasn1` utility. For information on the `AtoB` utility see, "ASCII to Binary Tool" on page 1189 and for the `dumpasn1` utility see, "dumpasn1 Tool" on page 1195.

Here's how you would do this verification:

1. Go to this directory: `<server_root>/bin/cert/tools`
2. Enter this command: `AtoB <input_file> <output_file>`, substituting `<input_file>` with the path to the file that contains the base-64 encoded data in ASCII format (from Step 2) and `<output_file>` with the path to the file to write the base-64 encoded data in binary format.
3. Next, enter this command: `dumpasn1 <ouput_file>`, substituting `<output_file>` with the path to the file to that contains the base-64 encoded data in binary format. Your output should look similar to this:

```
0 30   76: SEQUENCE {
   2 30   46:   SEQUENCE {
   4 06    3:     OBJECT IDENTIFIER extKeyUsage (2 5 29 37)
   9 01    1:     BOOLEAN TRUE
  12 04   36:     OCTET STRING
           :       30 22 06 05 2A 83 45 04 03 06 0A 51 82 E4 42 83
           :       33 93 DE 5F 35 06 06 2D 82 57 22 CD 09 06 05 51
           :       38 81 6A 4A
           :     }
  50 30   26:   SEQUENCE {
  52 06    3:     OBJECT IDENTIFIER issuerAltName (2 5 29 18)
  57 04   19:     OCTET STRING
           :       30 11 A4 0F 30 0D 31 0B 30 09 06 03 55 04 06 13
           :       02 55 53
           :     }
           :   }

0 warnings, 0 errors.
```

4. If the output doesn't appear right, repeat Step 1 through Step 3 to get the correct output.

Step 4. Copy the base-64 encoded blob in Step 2 (the output generated by the ExtJoiner) to the wizard screen and generate the certificate or the certificate signing request (CSR), if submitting the request to another CA..

Step 7. Copy the Certificate Signing Request

Based on the information you've entered in the previous steps, the wizard now displays the certificate signing request (CSR).

The request is in a base-64 encoded PKCS #10 format and is bounded by the marker lines `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`.

An example is show below:

```
-----BEGIN NEW CERTIFICATE REQUEST-----

MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBC6SAwHgYDVQQKExdOZXRzY2FwZSBDb21td
W5pY2F0aW9uczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFoXDTk5M
DIyMzE5MDAwMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQ
LEwZQZW9wbGUxFzAVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0e
TEjMCEGCSqGSIb3DbndgJARYUc3Vwcml5Yhvfggsvwryw4y7214vAOBgNVHQ8BAf8EBAMCBLAwFAY
JYIZIAYb4QgEBAQHBAQDAgCAMA0GCSqGSIb3DQEBBAUAA4GBAFi9FzyJlLmS+kzsue0kTXawbwamG
```

```
dYql2w4hIBgdR+jWeLmD4CP4xzmKdvQ6IqD2q8DBs9lRQu9JYg129oaCLpZfMNTPnMc3WPKO2pWZW
Um7waHEtdbo9vSpbJkXTM/2GhWbsO5vLdeOxrPGxihkHgV/
vmqCl4HW7AorqGgyfygbhgthgutrhryj

-----END NEW CERTIFICATE REQUEST-----
```

Do not modify the CSR; you must send it to the CA as it is.

Copy the CSR, including the marker lines `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`, to a text file. If you are running the wizard on a Windows NT system, you can also copy the CSR to the Windows clipboard. In a Unix system, you may have to open an application, such as Netscape Composer, with a clipboard.

The wizard also copies the CSR to a text file it creates in the configuration directory, which is located at

`<server_root>/cert-<instance_id>/config`

The name of the text file varies depending on for which key pair you generated the CSR:

- If the CSR is for a Certificate Manager CA signing certificate, the file name will be `cacsr.txt`.
- If the CSR is for a Registration Manager signing certificate, the file name will be `racsr.txt`.
- If the CSR is for a Data Recovery Manager transport certificate, the file name will be `kracsr.txt`.
- If the CSR is for a SSL server certificate, the file name will be `sslcsr.txt`.

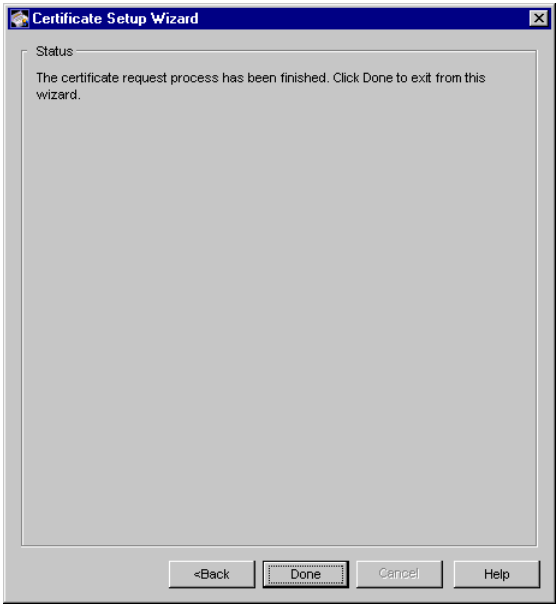
Step 8. Check the Certificate Request Status

The wizard now informs you of the status of the request.

- If you requested a self-signed CA certificate, the wizard automatically submits the CSR to the CA. If the CSR includes all the required information, the CA signs the certificate and returns it to the wizard, which then installs it in the appropriate token.
- If you requested any other certificate, you must submit the CSR to a CA of your choice manually. This is explained in Step 9.

Step 9. Send the Certificate Signing Request to a CA

This step in not part of the wizard process. Follow these instructions to manually send the CSR to a CA of your choice. The CA can be internal or external.



- An internal CA is any CA deployed internally by your organization—for example, a Certificate Manager installed in the same or another instance of Certificate Management System. See “Sending the CSR to an Internal CA” on page 258.
- An external CA is any public CA. Before sending the CSR to a public CA, make sure that the CA can issue the certificate you want to request. Also, it is a good idea to read the policy statement published by a CA to see whether the CA imposes any restrictions on the validity period or usage of the certificate. See “Sending the CSR to an External CA” on page 259.

Sending the CSR to an Internal CA

The following instructions assume that your internally deployed CA is a Certificate Manager and that you are using the default HTML forms provided for end-entity enrollment. If you have customized these forms, you should follow the appropriate instructions.

To send the CSR manually to an internal CA:

1. Locate the text file to which you copied the CSR earlier (see "Step 7. Copy the Certificate Signing Request" on page 255).

2. Open a web browser window.

3. Enter the URL to the CA's home page.

   By default, the CA's home page is the end entity services interface. Depending on the port at which the CA is listening to end-entity requests (see "End-Entity Ports" on page 158) the URL to the end entity services is one of the following:

   `http://<host_name>:<end_entity_port>`

   or

   `https://<host_name>:<end_entity_port>`

   where `<host_name>` is in the form `<machine_name>.<your_domain>.<domain>`

   The end entity services interface appears.

4. Click the Enrollment tab.

5. In the menu list, click the appropriate link:

   - If the CSR is for a subordinate CA certificate, select the manual enrollment link in the Certificate Manager Enrollment section.
   - If the CSR is for a Registration Manager's signing certificate, select the manual enrollment link in the Registration Manager Enrollment section.
   - If the CSR is for an SSL server certificate, select either the manual or the directory-based link in the Server Enrollment section.

   For information on how the default server enrollment forms work, see "Certificate Issuance to Servers" on page 1081.

6. In the form that appears, enter the required information and paste the CSR from the text file.

   Be sure to include the marker lines, `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`.

7. Submit the request.

8. When the CA sends you a response, save the information in a text file for future reference or inquiry.

   - If you submitted the request using a manual enrollment form, the request gets queued.

     An agent needs to process and approve the certificate request, which the CA signs then and delivers back to the email address specified in the request. You can contact the CA agent to find out when the certificate will be delivered to you.

   - If you submitted the request using an automated enrollment form, the request gets processed automatically by the CA.

     Keep in mind that the CA can reject or queue the request if it fails any of the authentication or policy checks. A request that gets queued must be processed manually by an agent.

9. When you receive the certificate from the CA, install it following the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260.

Sending the CSR to an External CA

To send the CSR manually to an external CA:

1. Locate the text file to which you copied the CSR (see "Step 7. Copy the Certificate Signing Request" on page 255).
2. Open a web browser window.
3. Navigate to the CA's home page by entering the appropriate URL in the browser window.
4. Locate the form that allows you to submit certificate requests for servers.
5. Enter the required information and paste the CSR from the text file.

   Be sure to include the marker lines, `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----`.
6. Submit the request.
7. When the CA sends you a response, save the information in a text file for future reference or inquiry.

8. When you receive the certificate from the CA, install it following the instructions in "Using the Wizard to Install a Certificate or Certificate Chain" on page 260.

Using the Wizard to Install a Certificate or Certificate Chain

The Certificate Setup Wizard allows you to install or import the following certificates into either an internal or external token used by the currently selected CMS instance:

- Any of the certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager installed in the currently selected CMS instance
- Any other trusted CA certificates (certificates of CAs that you want to trust)
- Certificate chains

  A certificate chain typically includes a collection of certificates: the subject certificate, the trusted root CA certificate, and any intermediate CA certificates needed to link the subject certificate to the trusted root. However, the certificate chain the wizard allows you to import must include only CA certificates; none of the certificates can be a user certificate.

  In a certificate chain, each certificate in the chain is encoded as a separate DER-encoded object. When the wizard imports a certificate chain, it imports these objects one after the other, all the way up the chain to the last certificate, which may or may not be the root CA certificate. If any of the certificates in the chain already exist in the local certificate database, the wizard replaces them by the ones included in the chain. If the chain includes intermediate CA certificates, the wizard adds them to the certificate database as *untrusted* CA certificates.

The certificate or certificate chain you provide to the wizard for installation must be in one of the data formats supported by the wizard. This is explained in "Data Formats for Installing Certificates and Certificate Chains" on page 261.

Using the wizard to install a certificate or certificate chain involves the following steps, described in detail on page 263:

- Step 1. Select the Operation
- Step 2. Select the Certificate or Certificate Chain
- Step 3. Specify the Location of the Certificate
- Step 4. View the Certificate or Certificate Chain
- Step 5. Install the Certificate or Certificate Chain
- Step 6. Verify the Certificate Status

Data Formats for Installing Certificates and Certificate Chains

The wizard can accept certificates and certificate chains in several data formats. This section briefly explains the data formats recognized by the wizard.

Binary Formats

The wizard can recognize certificates and certificate chains in the following binary formats:

- DER-encoded certificate

  This is a single binary DER-encoded certificate.

- PKCS #7 SignedData objects

  This is a `PKCS #7 SignedData` object. The only significant field in the `SignedData` object is the certificate. In particular, the signature and the contents are ignored. The PKCS #7 format allows multiple certificates to be downloaded at once.

- DER-encoded certificates

  These are DER-encoded certificates that may or may not be wrapped in a base-64 encoding package surrounded by the delimiters `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`.

- Netscape Certificate Sequence

This is a simpler format for downloading certificate chains. It consists of a `PKCS #7 ContentInfo` structure, wrapping a sequence of certificates. The value of the `contentType` field should be `netscape-cert-sequence`, while the content field is the following structure:

```
CertificateSequence ::= SEQUENCE OF Certificate
```

This format allows multiple certificates to be downloaded at once.

Text Formats

The wizard can also import certificates and certificate chains in text formats. Here's what you should be aware of when using the wizard to install a certificate or certificate chain in text format:

The text format must begin with the following line:

```
-----BEGIN CERTIFICATE-----
```

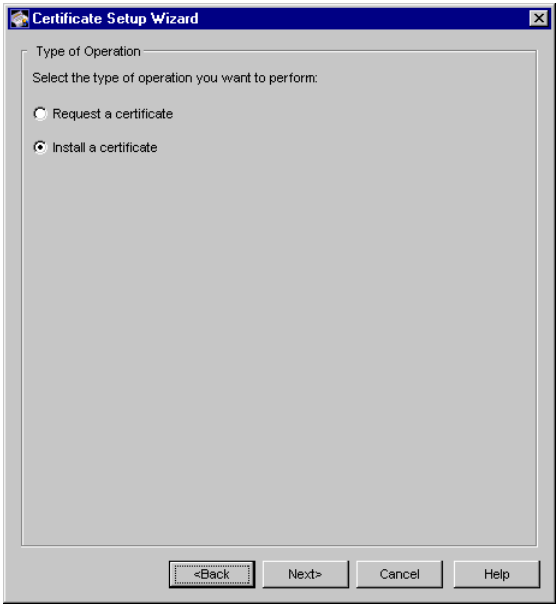
Following this line should be the certificate data, which can be in any of the binary formats described in "Binary Formats" on page 261. This data should be base-64 encoded as described by RFC 1113 (for details, see `http://www.scit.wlv.ac.uk/rfc/rfc11xx/RFC1113.html`).

Following the certificate data must be this line:

```
-----END CERTIFICATE-----
```

Step 1. Select the Operation

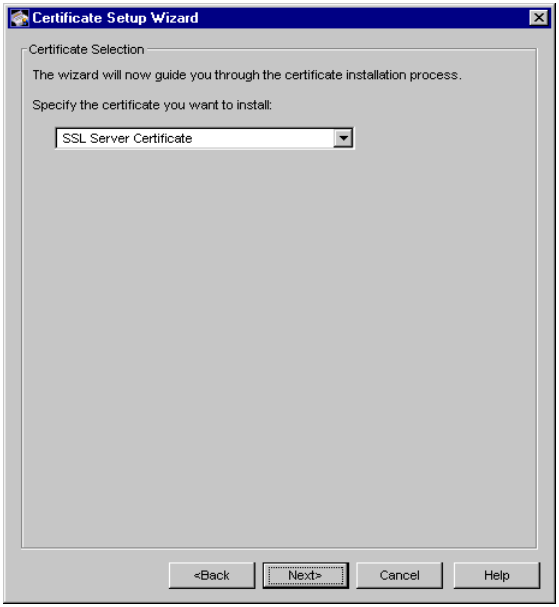
Indicate whether you want to request a certificate or install a certificate.



For the sake of completing the instructions that follow, assume that you chose to install a certificate.

Step 2. Select the Certificate or Certificate Chain

Select the certificate you want to install.



The drop-down list shows various options:

- Any of the certificates used by the subsystems—Certificate Manager, Registration Manager, or Data Recovery Manager—installed in the currently selected CMS instance.
- Any other trusted CA certificates
- Certificate chains

Depending on whether you want to install a CMS certificate, any other trusted CA certificate, or a CA certificate chain, choose the appropriate option from the list box:

- Certificate Manager Signing Certificate—choose this option if you want to install a signing certificate for the Certificate Manager installed in the currently selected CMS instance.

- Registration Manager Signing Certificate—choose this option if you want to install a request signing certificate for the Registration Manager installed in the currently selected CMS instance.
- Data Recovery Manager Transport Certificate—choose this option if you want to install a transport certificate for the Data Recovery Manager installed in the currently selected CMS instance.
- SSL Server Certificate—choose this option if you want to install an SSL server certificate for the Certificate Manager, Registration Manager, or Data Recovery Manager installed in the currently selected CMS instance.
- Server Certificate Chain—choose this option if you want to install a CA certificate chain; the CA certificate will be included in the chain.
- Other Trusted CA Certificate—choose this option if you want to install a trusted CA certificate.

Step 3. Specify the Location of the Certificate

Locate the certificate or certificate chain you want to install.

You can keep the certificate or certificate chain in a text file or copy it to the text area on the wizard screen. Here is some information that will help you decide on the location.

- Keeping the certificate or certificate chain in a text file

  The wizard can import a certificate or certificate chain from a text file in *text* as well as *binary* formats; see "Data Formats for Installing Certificates and Certificate Chains" on page 261. If you have copied the certificate or certificate chain to a text file, you will be required to provide the wizard with the absolute path to that file. The file must be located in the host system the wizard is running. If the file is located elsewhere, exit from the wizard, copy the file to the local disk, and restart the wizard.

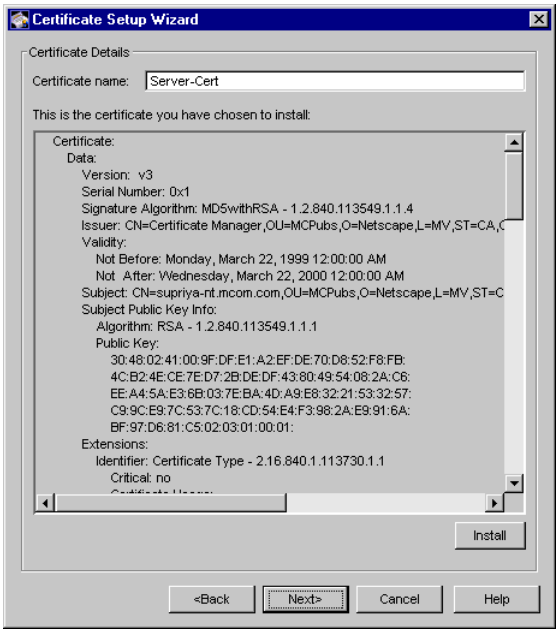
- Copying the certificate or certificate chain to the text area on the wizard screen

  You can paste the certificate or certificate chain into the text area provided by the wizard. This is a text input field, so you can paste the certificate or certificate chain in text format only. For example, if you are installing a certificate, it base-64 encoded certificate blob should look similar to this:

```
-----BEGIN CERTIFICATE-----

MIICKzCCAZSgAwIBAgIBAzANgkqkiG9w0BAQQFADA3MQswCQYDVQQGEwJVUzERMA8GA1UEChMITmV
0c2NhcGUxFTATBgNVBAsTDFN1cHJpeWEncyBDQTAeFw05NzEwMTgwMTM2MjVaFw05OTEwMTgwMTM2
MjVaMEgxCzAJBgNVBAYTAlVTMREwDwYDVQQKEwhOZXRzY2FwZTENMAsGA1UECxMEUHawczEXMBUGA
1UEAxMOU3Vwcml5YSBTaGV0dHkwgZ8wDQYJKoZIhdfNAQEBBQADgY0AMIGJAoGBAMr6eZiPGfjX3u
RJgEjmKiqG7SdATYzBcABu1AVyd7chRFOGD3wNktbf6hRo6EAmM5R1Askzf8AW7LiQZBcrXpc0k4d
u+2j6xJu2MPm8WKuMOTuvzpo+SGXelmHVChEqooCwfdiZywyZNmgaMa2MS6pUkfQVAgMBAAGjNjA0
MBEGCWCGSAGG+EIBAQQEAwIAgDAfBgNVHMEGDAWgBTy8gZZkBhHUfWJM1oxeuZc+zYmyTANBgkqhk
iG9w0BAQQFAAOBgQBtI6z07Z635DfzX4XbAFpjlRlAYwQzTRYx8GfcNAqCqCwaSDKvsujvwbf91o3
j3UkdGYpcd2cYRCgKi4MwqdWyLtpuHAH18hHZ5uvi00mJYw8W2wUOsY0RCaIDy84hW3WWehBUqVK5
SYzJ4oTjx7dwNMdGwbWfpRqjd1A==thghtelirhe395uy85ifnf8HUYUHU

-----END CERTIFICATE-----
```

Step 4. View the Certificate or Certificate Chain

The wizard displays the certificate or certificate chain you have chosen to install. Make sure you have chosen the right one; otherwise, use the Back button to go back and locate the right one. Specify a nickname for the certificate.



Step 5. Install the Certificate or Certificate Chain

The wizard shows the certificate or certificate chain information you have selected for installing. You should check the information to make sure that you have chosen the correct one for installing.

After verifying that the certificate you have chosen is the correct one, click the Install button. The wizard installs the certificate or the CA chain in the token you have chosen.

- If you installed a certificate that has been issued by CA whose certificate chain doesn't exist in the certificate database, you must add that CA's certificate chain to the database. To add the CA chain to the database, copy the CA chain to a text file, start the wizard again, and install the CA chain.

- If you installed (or imported) a certificate chain, the wizard adds (to the local trust database) the first certificate in the chain as a trusted CA certificate and any subsequent certificates as untrusted CA certificates. For more information on how the wizard installs a certificate chain, see "Using the Wizard to Install a Certificate or Certificate Chain" on page 260.

Step 6. Verify the Certificate Status

This step is applicable only if you installed a certificate chain.

After you install a certificate chain in the trust database of a CMS instance, check the trust status of each certificate that got installed, and make sure that the correct CA certificates are trusted. For instructions, see "Changing the Trust Settings of a CA Certificate" on page 299.

Configuring the Server's Security Preferences

Configuring the server's security preferences involves identifying the following:

- The SSL server certificates the server must use for authenticating to the end entity, agent, and administration interfaces. For details, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269.
- The SSL client certificate the server must use for authenticating to the publishing directory. For details, see "Getting an SSL Client Certificate for a Subsystem" on page 271.
- The version of SSL that an instance of Certificate Management System must use during SSL communication. The latest version is SSL version 3, but many older clients use SSL version 2. Because client authentication is required for performing privileged operations, you must enable SSL version 3 ciphers supported by Certificate Management System. For details, see "Setting Up Cipher Preferences for SSL Communications" on page 273.

Configuring the Server to Use Separate SSL Server Certificates

You can configure a CMS instance to use separate SSL server certificates for authenticating to Netscape Console, the Agent Services interface, and the end entity services interface.

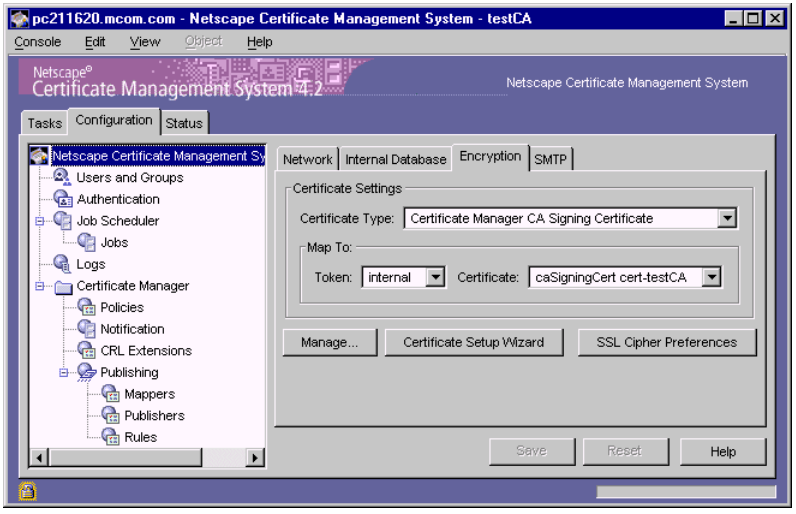
This configuration involves the following steps:

- Step 1. Get the Required SSL Server Certificates
- Step 2: Update the Configuration

Step 1. Get the Required SSL Server Certificates

You must first request and install the required number of SSL server certificates for the particular CMS instance. For instructions, see "Getting New Certificates for the Subsystems" on page 277.

Once you have installed the certificates, you should be able to see them in the list of SSL server certificates in the Encryption tab of the CMS window.



Step 2: Update the Configuration

After you verify that the certificates are installed, configure the server as follows:

1. Stop the CMS instance; see "Stopping Certificate Management System" on page 137.
2. Open the configuration file (`CMS.cfg`) in a text editor; to locate the file, see "Locating the Configuration File" on page 85.
3. Change the configuration:
   - To change the certificate used for authenticating to the Agent Services interface, locate the `agentGateway.https.nickName` parameter and change its value to the nickname of the new SSL server certificate.

     For example, if the nickname of the SSL server certificate is `SSLServerCert_agt`, the configuration should look like this:

     ```
     agentGateway.https.nickName=SSLServerCert_agt
     <instance_id>
     ```

   - To change the certificate used for authenticating to the end-entity services interface, locate the `eeGateway.https.nickName` parameter and change its value to the nickname of the new SSL server certificate.

     For example, if the nickname of the SSL server certificate is `SSLServerCert_ee`, the configuration should look like this:

     ```
     eeGateway.https.nickName=SSLServerCert_ee <instance_id>
     ```

   - To change the certificate used for authenticating to the administration interface, Netscape Console, locate the `radm.https.nickName` parameter and change its value to the nickname of the new SSL server certificate.

     For example, if the nickname of the SSL server certificate is `SSLServerCert_admin`, the configuration should look like this:

     ```
     radm.https.nickName=SSLServerCert_admin <instance_id>
     ```
4. Save your changes.
5. Start the server; see "Starting Certificate Management System" on page 128.

Getting an SSL Client Certificate for a Subsystem

By default, the Certificate Manager uses its SSL server certificate for SSL client authentication to the publishing directory. If you want the subsystem to use other certificate for authenticating to the publishing directory, you can do so. This section explains how to get an SSL client certificate for a subsystem and how to configure the subsystem to use that certificate for authenticating to the publishing directory.

Getting an SSL client certificate for a subsystem involves the following steps:

- Step 1. Generate a Key Pair for the Subsystem
- Step 2. Generate a Certificate Signing Request for the Key Pair
- Step 3. Submit the CSR to the CA
- Step 4. Ask an Agent to Approve the Request
- Step 5. Install the Certificate in the Internal Database
- Step 6. Configure the Subsystem to Use This Certificate

Step 1. Generate a Key Pair for the Subsystem

Generate a key pair for the subsystem following the instructions in "Generating a New Key"; see Appendix E, "Key Database Tool" in the online version of this document.

Step 2. Generate a Certificate Signing Request for the Key Pair

Generate a certificate signing request (CSR) for the key pair following the instructions in "Creating a Certificate Request"; see Appendix D, "Certificate Database Tool" in the online version of this document. Be sure to include your email address in the request so that the CA can deliver the certificate to you.

Step 3. Submit the CSR to the CA

Submit the CSR to the CA. Because the request needs some modifications, be sure to use the *manual* enrollment process; this way you can tell the person who will process your request about the changes he or she must make before approving the request. For example, if you are submitting the request to a Certificate Manager (CA), be sure to use the *manual* enrollment form; do not use the automated enrollment forms, such as the directory-based server enrollment form. Include your email address in the request so that the CA can deliver the certificate to you.

Step 4. Ask an Agent to Approve the Request

If you submitted the request to a Certificate Manager (CA), ask its agent to process the request. The agent must make sure that only the SSL Client option for certificate type is selected in the request, and then approve it. (For certificates with no Netscape Certificate Type extensions, the key usage extension must be included with *signing* and *encryption* bits set.)

If you submitted the request to any other CA, you must ask the person managing that CA to make the same changes to the request before approving it.

Step 5. Install the Certificate in the Internal Database

When you receive the request, follow the instructions in "Adding a Certificate to the Database" (see Appendix D, "Certificate Database Tool" in the online version of this document) and install the certificate in the token you used to generate the key pair.

Step 6. Configure the Subsystem to Use This Certificate

After you install the certificate, configure the Certificate Manager to use the new certificate for SSL client authentication to the publishing directory. For instructions, see "Step 5. Identify the Publishing Directory" on page 830.

Setting Up Cipher Preferences for SSL Communications

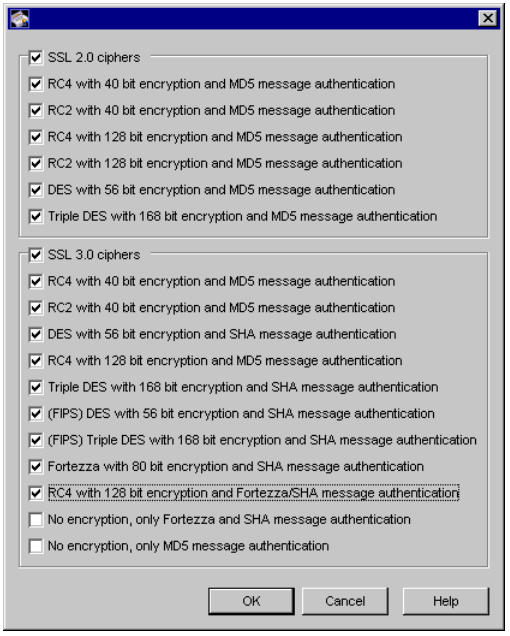
A *cipher* is the algorithm used in encryption. Some ciphers have *stronger* encryption capabilities than others. Generally speaking, the more bits a cipher uses during encryption, the harder it is to decrypt the data.

When a client initiates an SSL connection with Certificate Management System, it lets the server know what ciphers it prefers to use to encrypt information. In any two-way encryption process, both parties must use the same ciphers. A number of ciphers are available; your server needs to be able to use the most popular ones.

SSL Ciphers Supported in Certificate Management System

Figure 8.1 shows the ciphers supported by Certificate Management System (on the server side). The figure shows SSL 2.0 and 3.0 ciphers supported in the domestic (US and Canada) version of Certificate Management System. Note that Certificate Management System has received *retail* status from the United States Department of Commerce Bureau of Export Administration; under new regulations, retail status makes it possible to export Certificate Management System with the same encryption and cryptographic features available in the US and Canada. For more information, see Appendix E, "Export Control Information" of *Netscape Certificate Management System Installation and Deployment Guide.*

Figure 8.1 SSL version 2.0 and 3.0 cipher suites supported (in the domestic version)



You can choose ciphers from the SSL 2.0 protocol, as well as from SSL 3.0. To specify which ciphers your server can use, check them in the list of ciphers to enable them. Unless you have a compelling reason not to use a specific cipher, you should check them all, except as noted in the warning that follows. For a detailed description of ciphers, see "Ciphers Used with SSL" in Appendix E of *Managing Servers with Netscape Console*.

Warning You might not want to check the options that say "No Encryption, only MD5 message authentication" and "No Encryption, only Fortezza and SHA message authentication." The reason for this is, if no other ciphers are available on the client side, the server will use these and no encryption will occur.

Previous US law prohibited the export of software with strong encryption, so most browsers still in use outside of the US and Canada do not support 128-bit encryption. Disabling all 40-bit ciphers will ensure that all connections use higher-grade security, but will prevent access to your service to many users outside of the US and Canada.

Note that Netscape Communicator has received *retail* status from the United States Department of Commerce Bureau of Export Administration; under new regulations, retail status makes it possible to export Communicator with the same encryption and cryptographic features available in the US and Canada.

Prior to the retail status, international users of Netscape Communicator (with encryption capability restricted to 40-bit encryption) could use Netscape's International Step-Up program to *step up* to stronger encryption, 56-bit, 128-bit, or 168-bit. Step-up refers to the ability of export browsers to establish strong SSL sessions with domestic SSL servers, if they have the appropriate step-up certificates.

You can find general information on the International Step-Up program and on getting step-up certificates at this URL:

```
http://home.netscape.com/products/security/technology/
128bit.html
```

Information on configuring clients and servers for the international step-up feature (after obtaining step-up certificates) is available at this URL:

```
http://developer.netscape.com/tech/security/stepup/
overview.html
```

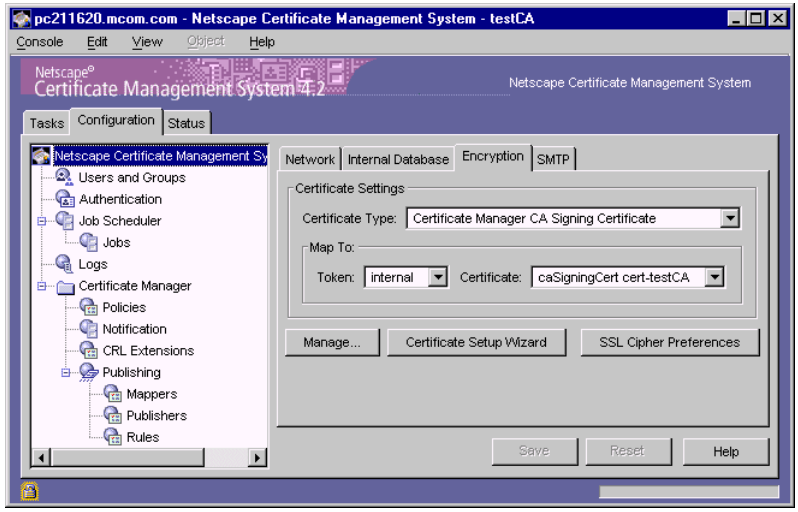
Configuring the Server to Use Specific Ciphers

You can set a number of systemwide preferences for SSL by specifying the ciphers that Certificate Management System should recognize and use during SSL communication; the server applies the cipher settings you choose to all the SSL (HTTPS) ports it uses.

To change the cipher settings for a CMS instance:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

2. Select the Configuration tab, and then in the right pane, select the Encryption tab.



3. Click SSL Cipher Preferences, and choose the appropriate options.

   For details, see "Setting Up Cipher Preferences for SSL Communications" on page 273.

4. Click OK.

   You are returned to the Encryption tab.

5. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Getting New Certificates for the Subsystems

You may need to get new certificates for Certificate Management System. Getting a new certificate means getting a certificate based on a new public and private key pair.

The sections that follow explain how to get new certificates for the Certificate Manager, Registration Manager, and Data Recovery Manager using the Certificate Setup Wizard. Alternatively, you can use the command-line utility called the *Certificate Database Tool*; for details about this utility, see Appendix D, "Certificate Database Tool" in the online version of this document.

Getting a new certificate involves the following steps:

- Step 1. Plan for the New Certificate
- Step 2. Request the New Certificate
- Step 3. Install the New Certificate
- Step 4. Deploy the New Certificate

Step 1. Plan for the New Certificate

Getting a new certificate for a subsystem requires careful planning. This section provides some guidelines that will help you request and install the new certificate.

Determine which certificate you want to get

You can get CA signing and SSL server certificates for the Certificate Manager; signing and SSL server certificates for the Registration Manager; and transport and SSL server certificates for the Data Recovery Manager.

- If you have deployed a Certificate Manager as your root CA and if you want to get a new self-signed CA certificate for that Certificate Manager, you must consider the possible effects on your PKI setup of changing the key pair of the root CA. If you reissue the Certificate Manager's CA signing certificate with a new key material, none of the certificates issued or signed by the CA using its old key will work; the reason for this is, when you change the root CA key, all certificates that rely on the CA certificate for validation will no

longer be validated. For example, if the CA has issued certificates to subordinate Certificate Managers, Registration Managers, Data Recovery Managers, and agents, all those certificates will become invalid—the Certificate Managers (subordinate), Registration Managers, and Data Recovery Managers will fail to function, and agents will fail to access agent interfaces.

Before getting a new self-signed certificate for the Certificate Manager, therefore, you must address issues involved in deploying the new root CA certificate across your enterprise. It is beyond the scope of this document to explain how you should deploy the new CA certificate. You may find it useful to go over some of the deployment issues discussed in the document available at this URL:

`http://help.netscape.com/kb/corporate/19980710-25.html`

- If you have deployed a Certificate Manager as a subordinate CA (that's chained to a root CA) and if you want to get a new subordinate CA certificate for that Certificate Manager, you must consider the possible effects on your PKI setup of changing the key pair of the subordinate CA. When you change the subordinate CA key, all certificates that rely on the subordinate CA certificate for validation will no longer be validated. Before getting a new subordinate certificate, therefore, you must plan to address issues involved in deploying the new subordinate CA certificate across you enterprise.

- If you want to get a new signing certificate for a Registration Manager, check whether the Registration Manager has been set up as a trusted manager for any other subsystems—that is, you must identify the subsystems that have been configured to receive requests from this Registration Manager; see "Trusted Managers" on page 181. You will need to replace the existing signing certificate with the new one in all these subsystems.

- If you want to get a new transport certificate for a Data Recovery Manager, you must identify the end-entity interfaces or forms that have been set up for the archival of end users' encryption private keys; see "How Key Archival Works" on page 1119. You will need to replace the existing transport certificate with the new one in all these forms.

- If you want to get a new SSL server certificate for a Certificate Manager, decide whether you want the subsystem to also use this certificate for SSL client authentication to the publishing directory. If so, you will have to request the certificate with the appropriate extensions set, and after installing the certificate you will have to configure the publishing directory.
- You can get any number of SSL server certificates.

Decide on the CA that will sign the certificate

If you want to get a new self-signed CA certificate, you don't have to make this decision, because the CA itself signs it. For all other certificates, you must decide on the CA that will sign the certificate.

If you want the certificate to be signed by an internally deployed CA, check to be sure that the CA can issue the certificate you want request.

If you want the certificate to be signed by a public CA, find out the following:

- Does the public CA have a public policy statement? If one is available, read it; it may help you decide whether to request the certificate from this CA.
- Is the public CA's certificate already installed in the trusted CA in the trust database of Certificate Management System? If not, do you want to install it?
- Is the public CA a trusted CA in the trust database of Certificate Management System? If not, do you want to trust it?
- Can the public CA issue the certificate you want to request?
- Does the public CA impose any restrictions on certificates it issues? For example, if you are planning for requesting a subordinate CA certificate for a Certificate Manager, you may want to find out whether the public CA imposes any restrictions on the validity period, volume, or type of certificates your CA can issue. If you are planning for requesting a signing certificate for a Registration Manager, you may want to find out whether the public CA imposes any restrictions on the validity period or the number of certificate requests the Registration Manager can sign using the certificate. If you are planning for requesting a transport certificate for a Data Recovery Manager, you may want to find out whether the public CA imposes any restrictions on the validity period or the number of keys the Data Recovery Manager can archive using the certificate.

- What information does the public CA expects you to provide with the certificate request?
- How long will the public CA take to deliver the certificate, and how will the certificate be delivered to you? (The most common delivery mechanism is by email.)

Determine the token for generating the key pair

Identify the token, internal or external, that you want to use to generate the key pair for the certificate and to store the certificate. If you want to use an existing token, you must know the password that protects the token. If the token is external, make sure that the token is installed properly; see "Installing External Tokens" on page 236.

Determine certificate formulation information

Decide on the subject DN and nickname for the certificate you want generate. Also decide on details such as the key algorithm, key size, extensions, and validity period for the certificate.

Step 2. Request the New Certificate

Once you have all the information, go ahead and request the certificate. The Certificate Setup Wizard built into the CMS window automates the process of requesting certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager. You can use this wizard to generate a new certificate request and submit the request to any CA for signing. For instructions, see "Using the Wizard to Request a Certificate" on page 243.

Step 3. Install the New Certificate

When you receive the certificate from the CA, you must install it in the token that contains the key pair for this certificate; it must be the token you used to generate the request in Step 2.

The Certificate Setup Wizard automates the process of installing certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager. You may use this wizard to install the new certificate. For instructions, see "Using the Wizard to Install a Certificate or Certificate Chain" on page 260.

Step 4. Deploy the New Certificate

In this step, follow the instructions appropriate for the certificate you installed:

- If you installed a new CA signing certificate for a Certificate Manager, see "Deploying Certificate Manager's CA Signing Certificate" on page 281.
- If you installed a new signing certificate for a Registration Manager, see "Deploying Registration Manager's Signing Certificate" on page 283.
- If you installed a new transport certificate for a Data Recovery Manager, see "Deploying Data Recovery Manager's Transport Certificate" on page 284.
- If you installed a new SSL server certificate, see "Deploying a Subsystem's SSL Server Certificate" on page 286.

Deploying Certificate Manager's CA Signing Certificate

If you reissued the Certificate Manager's CA signing certificate with a new key material, none of the certificates issued by the CA using its old key will work For example, if the CA has issued certificates to subordinate Certificate Managers, Registration Managers, Data Recovery Managers, and agents, all those certificates will become invalid—the Certificate Managers (subordinate), Registration Managers, and Data Recovery Managers will fail to function, and agents will fail to access agent interfaces.

To reinstate your PKI setup, first you should get an agent certificate from the new CA so that you can get access to the Certificate Manager's agent interface. Once you have access to this interface, you will be able to approve new certificate requests from entities such as Registration Managers, Data Recovery Managers, and agents.

To request an agent certificate from the new CA:

1. Go to this directory: `<server_root>/cert-<instance_id>/config`
2. Open the configuration file, `CMS.cfg`, in a text editor.
3. Locate the `agentGateway.enableAdminEnroll` parameter and change its value from `false` to `true`.

   The modified parameter should look like this:
   `agentGateway.enableAdminEnroll=true`
4. Save your changes and close the file.
5. Restart the server.
6. Open a web browser window.
7. Go to the Certificate Manager's agent interface.

   The URL is in this format: `https://<host_name>:<agent_port>`
8. Enter all the information and request a new certificate.

   If you need more information on getting the first agent certificate, see section "Administrator/Agent Certificate Enrollment" in Chapter 5 of *Netscape Certificate Management System Installation and Deployment Guide*.
9. Once you get the certificate, install it in your browser.
10. Access the Certificate Manager's agent interface using your new certificate.

Deploying Registration Manager's Signing Certificate

If you installed a new Registration Manager signing certificate, you must also install this certificate in the certificate database of all subsystems (Certificate Manager, Registration Manager, and Data Recovery Manager) that trust this Registration Manager.

Here's what you must do:

1. Install the new signing certificate in the subsystems' certificate databases.

   Because the Registration Manager uses its signing certificate for SSL client authentication to the subsystems, you must add the new signing certificate to the internal database of all subsystems that have been configured to receive requests from the Registration Manager.

   To add the new certificate to a subsystem's internal database:

   - Note the instance ID and host name of the Registration Manager for which you got the new signing certificate; this information will help you to identify the Registration Manager in a subsystem's list of privileged users.
   - Copy the new signing certificate, in base-64 encoded format, to a text file.
   - Add the new certificate to the individual subsystem's internal database following the instructions in "Changing a Privileged User's Certificate" on page 220. Repeat this step for all subsystems that receive requests from this Registration Manager.

2. Ensure that the CA that signed the Registration Manager's certificate is in the certificate database of the subsystem.

   When a Registration Manager does SSL client authentication using its new certificate, the subsystem, as a part of validating the certificate presented by the Registration Manager, checks its trust database for the CA (certificate) that signed the Registration Manager's new certificate. If the subsystem does not find the CA as a trusted CA in its trust database, it rejects the Registration Manager.

   For instructions on checking the trust database of a subsystem, see "Viewing the Certificate Database Content" on page 295.

   - If you don't find the CA certificate, add it to the database as a trusted CA. For instructions on adding a CA certificate to the trust database of a subsystem, see "Installing a New CA Certificate in the Certificate Database" on page 301.

- If you find the CA certificate, verify its trust status. If it is untrusted, change the status to trusted. For instructions on changing the trust setting of a CA certificate, see "Changing the Trust Settings of a CA Certificate" on page 299.

Deploying Data Recovery Manager's Transport Certificate

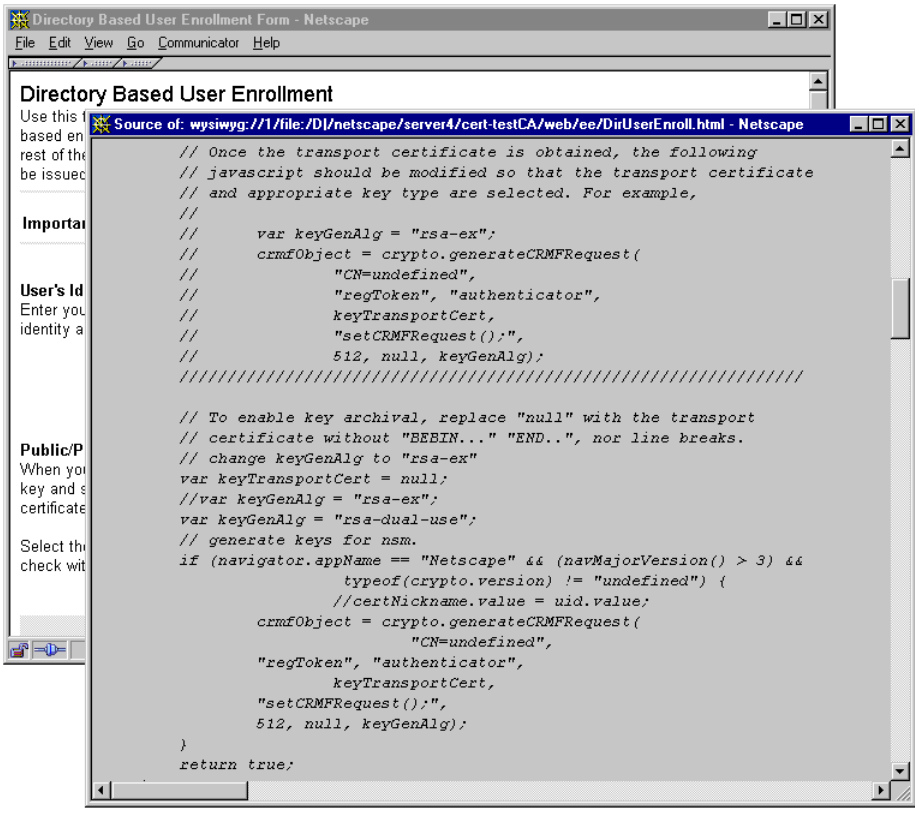
Because clients capable of generating dual key pairs use the transport certificate for encrypting end users' encryption private keys before sending them to the Data Recovery Manager, you must update the appropriate enrollment or key archival page to identify and use the new transport certificate. Otherwise, the Data Recovery Manager will fail to archive users' encryption private keys.

In general, here's what you need to do:

1. Locate the enrollment page that embeds the key archival feature.
2. View the HTML source, and identify the parameter that corresponds to the Data Recovery Manager's transport certificate.

   The default enrollment forms for end users embed this feature. Table 8.1 shows the default directory-based user enrollment form with the transport certificate-related information. (For more information, see "Step C. Customize the Certificate Enrollment Form" on page 1136.)

Table 8.1 Data Recovery Manager's transport certificate information in the enrollment form



3. Replace the current MIME-64 string with the one for the new transport certificate.

   To copy the MIME-64 string for the new transport certificate, locate the new transport certificate; the MIME-64 string for the certificate will be listed there.

4. Repeat steps 1 through 3 for any additional enrollment or key archival pages.

Deploying a Subsystem's SSL Server Certificate

By default, the Certificate Manager and Registration Manager use a single SSL server certificate to do server-side authentication to all the CMS ports. If a Certificate Manager is configured for SSL client authenticated communication with the publishing directory, it also uses its SSL server certificate for authenticating to the publishing directory. Depending on the purpose for which you requested this certificate, you should configure the server appropriately.

To configure the server to use this certificate for authenticating to one of the clients, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269. To configure the Certificate Manager to use this certificate for authenticating to the publishing directory, see "Step 5. Identify the Publishing Directory" on page 830.

Renewing Certificates for the Subsystems

All certificates used by Certificate Management System have a validity period beyond which they are not usable, unless the validity period is extended. For Certificate Management System to function properly, you must renew the certificates used by the Certificate Manager, Registration Manager, or Data Recovery Manager before they expire. For example, if you generated these certificates during CMS installation with a validity period of two years, at the end of which they will all expire; you must consider renewing them well in advance, maybe two months in advance.

When you renew a certificate, you get a new certificate with the same subject name and public and private key material as that of the existing certificate, but with an extended validity period.

The sections that follow explain how to renew certificates for the Certificate Manager, Registration Manager, and Data Recovery Manager using the Certificate Setup Wizard. Alternatively, you use the command-line utility called the *Certificate Database Tool*; for details about this utility, see Appendix D, "Certificate Database Tool" in the online version of this document.

Renewing an existing certificate involves the following:

- Step 1. Plan for Certificate Renewal
- Step 2. Renew the Existing Certificate

- Step 3. Install the Renewed Certificate
- Step 4. Deploy the Renewed Certificate
- Step 5. Restart the Server

Step 1. Plan for Certificate Renewal

Renewing a subsystem's certificate requires careful planning. This section provides some guidelines that will help you renew the certificate smoothly.

Before renewing a certificate:

- Note the subject DN and nickname of the certificate you want to renew.

  If you are planning on renewing the CA signing certificate of a Certificate Manager, make sure that the Certificate Manager has updated your LDAP directory with the most current certificate information (see "Publishing Certificates and CRLs to a Directory" on page 786).

  When you renew its CA signing certificate, the Certificate Manager automatically formulates a new certificate with the same public key and other details from the existing certificate, and publishes the new CA certificate to your LDAP directory.

- Identify the token, internal or external, that contains the keys for the certificate you want to renew. To use an existing token, you must know the password that protects the token. If the token is external, make sure that the token is installed properly; see "Installing External Tokens" on page 236.

- Decide on the validity period of the renewed certificate.

- Decide on the CA that will sign the certificate. If you want the certificate to be signed by a public CA, find out what information you need to provide with the certificate request. If you want the certificate to be signed by an internally deployed CA, check to be sure it can issue the certificate you want to request.

- Find out how long the CA will take to deliver the certificate to you. Make sure the renewed certificate is delivered to you well in advance so that you have a buffer period for installing and testing the renewed certificate, before the current certificate expires.

- Find out how the certificate will be delivered to you; the most common delivery mechanism is email. Make appropriate arrangements to receive the certificate.
- If you want to renew a subordinate CA certificate, plan how you will deploy the renewed CA certificate to end entities that rely on this certificate for validation.
- If you want to renew a root CA certificate, plan how you will deploy the renewed root CA certificate in your enterprise.

Step 2. Renew the Existing Certificate

Once you have all the information, go ahead and renew the certificate. The Certificate Setup Wizard built into the CMS window automates the process of renewing certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager. The wizard can generate a certificate request based on the existing key pair and submit the request to a CA for signing. For instructions on using the wizard, see "Using the Wizard to Request a Certificate" on page 243.

Important Be sure to use the existing key pair to generate the certificate signing request.

Also note that when you renew any of the CMS certificates—Certificate Manager's CA signing certificate, Registration Manager's signing certificate, Data Recovery Manager's transport certificate, and SSL server certificates—using the wizard, it saves the old certificate (in its base-64 encoded format) to a text file in the configuration directory, which is located here:

```
<server_root>/cert-<instance_id>/config
```

The name of the text file varies depending on the certificate you choose for renewal. If you choose to renew

- the Certificate Manager's CA signing certificate, it will be saved to a file named `prevCACert.txt.<timestamp>`.
- the Registration Manager's signing certificate, it will be saved to a file named `prevRACert.txt.<timestamp>`.
- the Data Recovery Manager's transport certificate, it will be saved to a file named `prevKRACert.txt.<timestamp>`.

- an SSL server certificate, it will be saved to a file named `prevSSLCert.txt.<timestamp>`.

The wizard also deletes the old certificate from the server's certificate database and adds the renewed certificate to the database, so that the server is able to use the renewed certificate upon restart. This feature restricts you to set the value of the `notBefore` attribute of the renewed certificate to either the current time or any time in the past, but not in the future.

If you set the validity period of the renewed certificate to begin on a future date and time, the server fails to use the certificate for its intended purposes. If this happens, you may either reinstall the old certificate (saved to the text file mentioned above) or renew the certificate again with an appropriate validity period.

Step 3. Install the Renewed Certificate

When you receive the renewed certificate from the CA, you must install it in the token that contains the key pair for the certificate; this is the token you used to generate the request in Step 2.

The Certificate Setup Wizard automates the process of installing certificates used by the Certificate Manager, Registration Manager, and Data Recovery Manager. For instructions on using the wizard, see "Using the Wizard to Install a Certificate or Certificate Chain" on page 260.

Step 4. Deploy the Renewed Certificate

In this step, you must follow the instructions appropriate for the certificate you installed:

- If you installed a renewed CA signing certificate for a Certificate Manager, see "Deploying Certificate Manager's Renewed CA Signing Certificate" on page 290.
- If you installed a renewed signing certificate for a Registration Manager, see "Deploying Registration Manager's Renewed Signing Certificate" on page 291.

- If you installed a renewed Data Recovery Manager transport certificate, see "Deploying Data Recovery Manager's Renewed Transport Certificate" on page 292.
- If you installed a new SSL server certificate, see "Deploying a Subsystem's Renewed SSL Server Certificate" on page 294.

Deploying Certificate Manager's Renewed CA Signing Certificate

If you installed a new CA signing certificate, deploy it in the PKI environment that depends on this certificate for validation. It is beyond the scope of this book to explain how you should deploy the new CA certificate. You may find it useful to go over some of the deployment issues discussed in the document available at this URL:

```
http://help.netscape.com/kb/server/980710-25.html
```

Deploying Registration Manager's Renewed Signing Certificate

Here's what you must do:

1. Install the renewed signing certificate in the subsystem's certificate database.

   Because the Registration Manager uses its signing certificate for SSL client authentication to the subsystems, you must add the renewed signing certificate to the internal database of all subsystems that have been configured to receive requests from the Registration Manager.

   To add the renewed certificate to a subsystem's internal database:

   - Note the instance ID and host name of the Registration Manager for which you got the signing certificate; this information will help you to identify the Registration Manager in a subsystem's list of privileged users.
   - Copy the renewed signing certificate, in its base-64 encoded format, to a text file.

- Add the renewed certificate to the individual subsystem's internal database following the instructions in "Changing a Privileged User's Certificate" on page 220. Repeat this step for all subsystems that receive requests from this Registration Manager.

2. Ensure that the CA that signed the Registration Manager's certificate is in the trust database of the subsystem.

   When a Registration Manager does SSL client authentication using its renewed certificate, the subsystem, as a part of validating the certificate presented by the Registration Manager, checks its trust database for the CA (certificate) that signed the Registration Manager's renewed certificate. If the subsystem does not find the CA as a trusted CA in its trust database, it rejects the Registration Manager.

   For instructions on checking the trust database of a subsystem, see "Viewing the Certificate Database Content" on page 295.

   - If you don't find the CA certificate, add it to the database as a trusted CA. For instructions on adding a CA certificate to the trust database of a subsystem, see "Installing a New CA Certificate in the Certificate Database" on page 301.

   - If you find the CA certificate, verify its trust status. If it is untrusted, change the status to trusted. For instructions on changing the trust setting of a CA certificate, see "Changing the Trust Settings of a CA Certificate" on page 299.

Deploying Data Recovery Manager's Renewed Transport Certificate

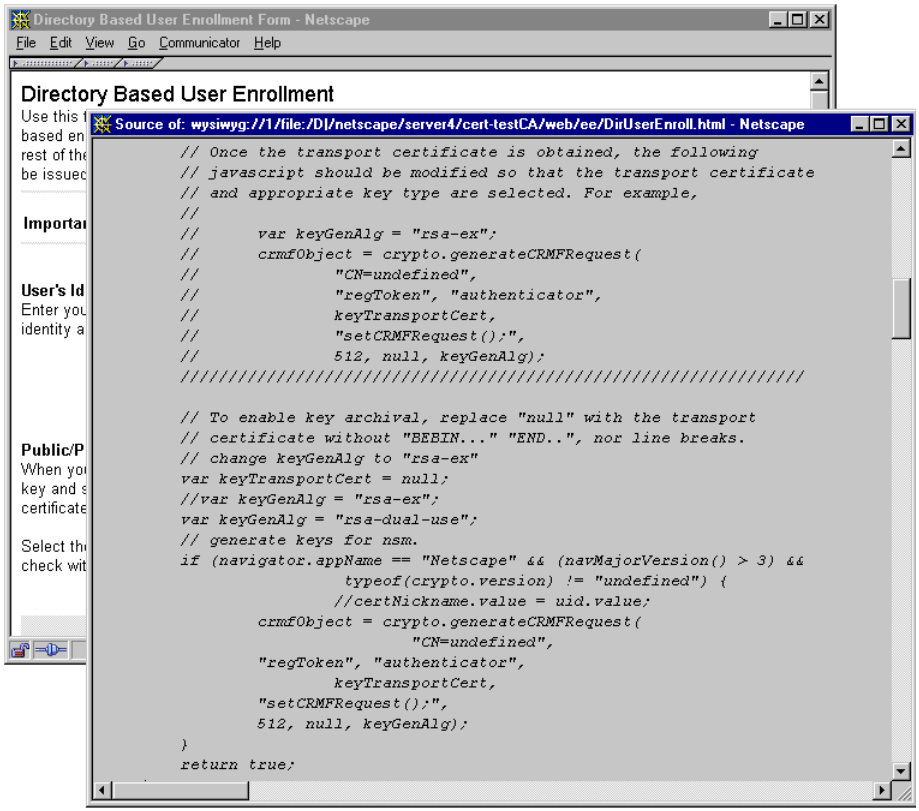
Because clients capable of generating dual key pairs use the transport certificate for encrypting end users' encryption private keys before sending them to the Data Recovery Manager, you must update the appropriate enrollment or key archival page to identify and use the renewed transport certificate. Otherwise, the Data Recovery Manager will fail to archive users' encryption private keys.

In general, here's what you need to do:

1. Locate the page that embeds the key archival feature.
2. View the HTML source, and identify the parameter that corresponds to the Data Recovery Manager's transport certificate.

   The default enrollment forms for end users embed this feature. Table 8.2 shows the default directory-based user enrollment form with the transport certificate-related information. (For more information, see "Step C. Customize the Certificate Enrollment Form" on page 1136.)

Table 8.2 Data Recovery Manager's transport certificate information in the enrollment form



3. Replace the current MIME-64 string with the one for the renewed transport certificate.

To copy the MIME-64 string for the renewed transport certificate, locate the certificate; the MIME-64 string for the certificate will be listed there.

4. Repeat steps 1 through 3 for any additional key archival or enrollment pages.

Deploying a Subsystem's Renewed SSL Server Certificate

By default, the Certificate Manager and Registration Manager use a single SSL server certificate to do server-side authentication to all the CMS ports. If a Certificate Manager is configured for SSL client authenticated communication with the publishing directory, it also uses the SSL server certificate for authenticating to the publishing directory. The Certificate Manager, if configured to function as a trusted manager to a Data Recovery Manager, also uses its SSL server certificate for SSL client authentication to the Data Recovery Manager. Depending on the purpose for which the certificate being renewed is used currently, you should configure the server appropriately.

- To configure the server to use this certificate for authenticating to one of the clients, see "Configuring the Server to Use Separate SSL Server Certificates" on page 269.
- To configure the Certificate Manager to use this certificate for authenticating to the publishing directory, see "Step 5. Identify the Publishing Directory" on page 830.

Step 5. Restart the Server

After you renew any of the CMS certificates using the wizard, you must restart the server. For instructions, see "Restarting Certificate Management System" on page 139.

Managing the Certificate Database

Each CMS instance has a certificate database (`cert7.db`), which is maintained in its internal token. This database contains certificates belonging to the subsystems installed in the CMS instance (see "Keys and Certificates for the Main Subsystems" on page 226) and various CA certificates the subsystems use for validating the certificates they receive.

Whether you use an internal token or an external token for generating and storing key pairs, Certificate Management System always maintains its list of trusted and untrusted CA certificates in its internal token.

You may need to add new certificates to the database, remove unwanted certificates from the database, or change the trust settings of CA certificates in the database. This section explains how to view the contents of the certificate database, delete unwanted certificates, and change the trust settings of CA certificates installed in the database using the CMS window. For information on adding certificates to the database, see "Certificate Setup Wizard" on page 242.

Note Certificate Management System also provides a command-line utility called `certutil` for managing its certificate database. See Appendix D, "Certificate Database Tool" in the online version of this document.

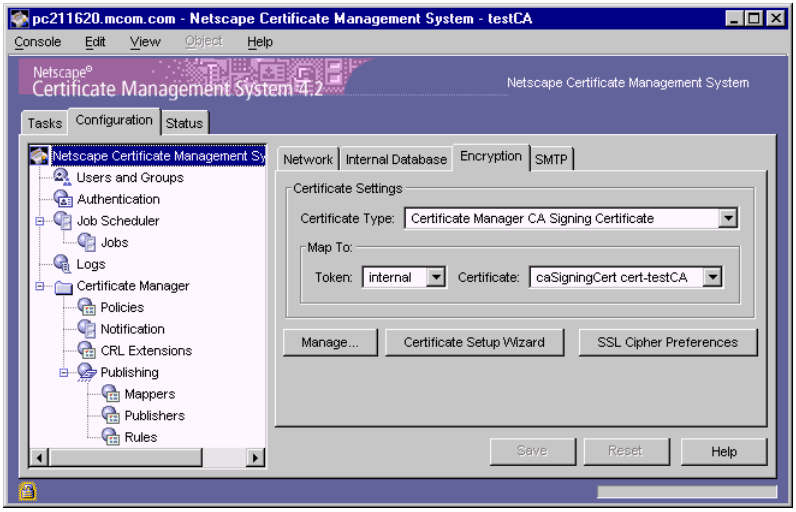
Viewing the Certificate Database Content

Each CMS instance has a certificate database that contains the list of certificates the server uses.

To view the contents of the database:

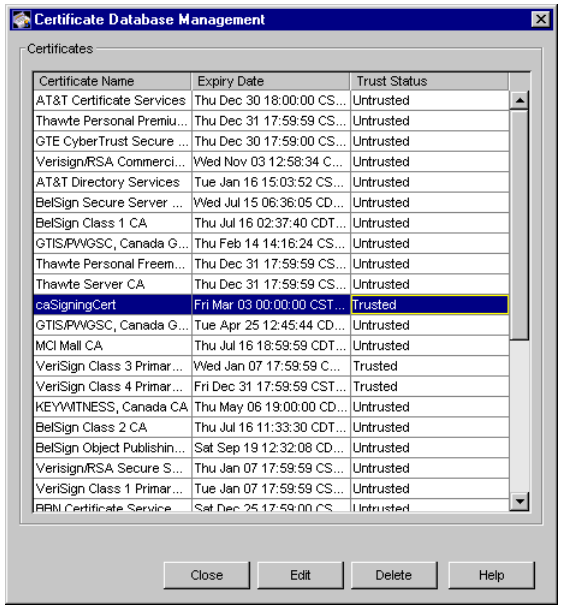
1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. Select the Configuration tab, and then in the right pane, select the Encryption tab.



3. Click Manage Certificate.

The Certificate Database Management window appears.



The window lists the certificates in a table, with each certificate occupying a row. The certificates are listed in alphabetical order. If the database contains multiple certificates with the same nickname, they are sorted by their validity periods; the most recently requested certificate is placed at the top.

For each certificate, you see the following information:

Certificate Name. Specifies the nickname of the certificate.

Expiry Date. specifies the date (and time) on which the certificate expires.

Trust Status. Specifies whether the CA is trusted or untrusted. To change the trust setting, see “Changing the Trust Settings of a CA Certificate” on page 299.

Deleting a Certificate From the Certificate Database

By default, the CMS certificate database includes a few public or third-party CA certificates. As an administrator, you should periodically check the contents of the certificate database and make sure that it doesn't include any unwanted CA certificates. For example, if the database includes CA certificates that you don't ever want to trust in your PKI setup, you should delete them.

Removing unwanted certificates also reduces the size of the certificate database.

Important When deleting CA certificates from the certificate database, be careful not to delete the *intermediate CA certificates*, which help a subsystem chain up to the trusted CA certificate. If in doubt, leave the certificates in the database as *untrusted* CA certificates; see "Changing the Trust Settings of a CA Certificate" on page 299.

To delete a CA certificate from the certificate database:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab, and then in the right pane, select the Encryption tab.
3. Click Manage Certificate.

   The Certificate Database Management window appears. The window lists all the certificates for the selected instance of Certificate Management System; the list is a table, with each certificate occupying a row.
4. Select the CA certificate you want to delete, and click Delete.
5. When prompted, confirm the delete action.
6. Click Close.

   You are returned to the Encryption tab.
7. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Changing the Trust Settings of a CA Certificate

Certificate Management System relies on the CA certificates in its certificate database for validating certificates it receives during an SSL-enabled communication. For example, when a Certificate Manager is authenticating a Registration Manager that has sent a certificate signing request, the Certificate Manager checks its certificate database to see whether the CA that has signed the certificate presented by the Registration Manager is included in the database as a *trusted* CA.

You may need to change the status of a currently trusted CA to untrusted (or vice versa) temporarily or permanently. For example, you may be notified that a CA is experiencing technical difficulty that prevents certificate authentication. By making the CA certificate untrusted, you can prevent entities whose certificates have been signed by that CA from successfully authenticating to Certificate Management System. You can then return the trust option to *trusted* when the CA notifies you that the problem has been resolved.

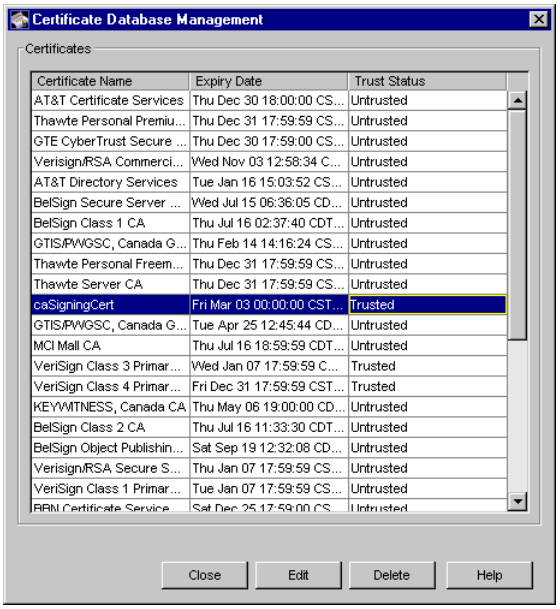
If you want to untrust a CA permanently, you should consider removing its certificate from the trust database altogether. For instructions, see "Deleting a Certificate From the Certificate Database" on page 298.

Changing the trust setting changes the trust flag (or bit) in the CA certificate.

To change the trust setting of a CA certificate:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab, and then in the right pane, select the Encryption tab.
3. Click Manage Certificate.

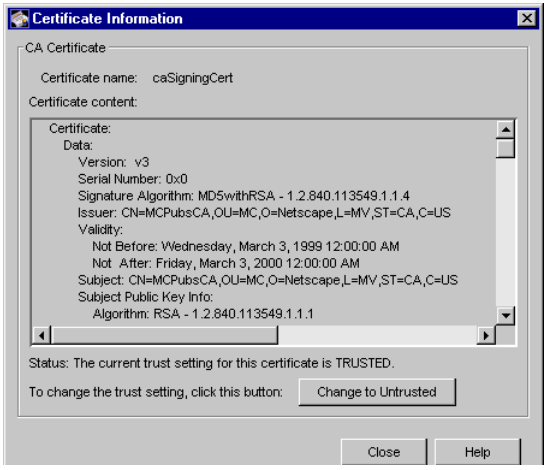
The Certificate Database Management window appears.



The window lists the certificates currently installed for the selected CMS instance; the list is a table, with each certificate occupying a row.

4. Select the CA certificate whose trust setting you want to modify, and click Edit.

   The Certificate Information window appears.



The window shows detailed information about the selected certificate, including serial number, validity period, subject name, issuer name, certificate fingerprint, and trust status.

If the certificate you selected is currently trusted, the window shows a button named "Change to Untrusted." If it is untrusted, the window shows a button named "Change to Trusted."

5. Click "Change to Untrusted" or "Change to Trusted," as appropriate.
6. Click Close.

   You are returned to the Certificate Database Management window. The certificate now shows a different trust status.
7. Click Close.

   You are returned to the Encryption tab.
8. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Installing a New CA Certificate in the Certificate Database

You may need to install new trusted CA certificates in the certificate database of a CMS instance. For example, assume that you renewed the signing certificate of a Registration Manager. Also assume that the CA that signed the Registration Manager's certificate is not included in the trust database of the Certificate Manager that has been configured to sign certificate requests from this Registration Manager.

When the Registration Manager attempts to request a service from the Certificate Manager (using the renewed certificate for SSL client authentication), the Certificate Manager fails to authenticate the Registration Manager. This happens because, as a part of validating the certificate presented by the Registration Manager, the Certificate Manager checks its certificate database for the CA that signed the Registration Manager's certificate. The Certificate Manager does not find the CA listed in its trust database as a trusted CA, so it rejects the Registration Manager's service request.

The Certificate Setup Wizard built into the CMS window automates the process of installing trusted CA certificates in the certificate database. For instructions on using the wizard, see “Using the Wizard to Install a Certificate or Certificate Chain” on page 260.

Important Be sure to choose the “Other Trusted CAs” option in Step 2 of the wizard process.

Installing a CA Certificate Chain in the Certificate Database

Any client or server software that supports certificates maintains a collection of trusted CA certificates in its certificate database. These CA certificates determine which other certificates the software can validate—in other words, which issuers of certificates the software can trust. In the simplest case, the software can validate only certificates issued by one of the CAs for which it has a certificate. It’s also possible for a trusted CA certificate to be part of a chain of CA certificates, each issued by the CA above it in a certificate hierarchy; for details on certificate hierarchies and certificate chains, see "How CA Certificates Are Used to Establish Trust" in Appendix D of *Managing Servers with Netscape Console*.

4

Authentication

Chapter

9

Introduction to Authentication

Authentication is the process of verifying the identity of a user that is requesting a service from Netscape Certificate Management System (CMS). More specifically, authentication involves acquiring and verifying the values of the configured attributes of the user. For example, the user might be prompted to log in with a user name and password, and then the server would look in a preconfigured database to verify that the user's password was correct.

Service requests to Certificate Management System come from any of the following users:

- End entities—general users or applications that make certificate issuance, renewal, and revocation requests
- Administrators—privileged users who connect to the server to do system or server administration tasks
- Agents—privileged users who connect to the server to do agent operations

This chapter explains how Certificate Management System identifies and authenticates these users, and it provides details about the various authentication methods supported by the server. After reading this chapter, you should be able to determine which of the authentication plug-in modules provided out of the box is suitable for your PKI deployment.

This chapter has the following sections:

Privileged-User Authentication

For authenticating privileged users, such as administrators and agents, Certificate Management System uses built-in authentication mechanisms.

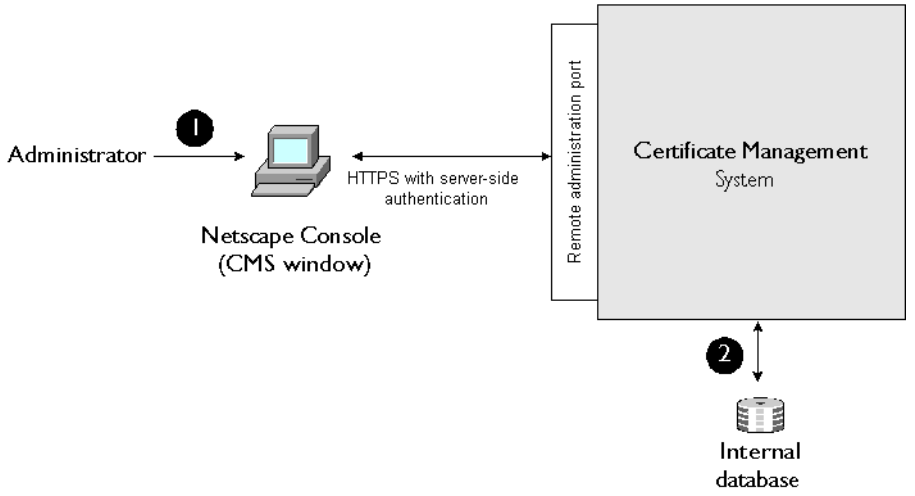
Authentication of Administrators

When an administrator makes an administrative request to Certificate Management System (from the CMS window within Netscape Console or from any command-line tool), the server needs to authenticate the administrator before processing the request. To facilitate this, Certificate Management System supports an authentication method that includes user ID- and password-based authentication from the client and SSL server authentication from the server.

Certificate Management System identifies and authenticates users with *administrator* privileges by checking their user IDs and passwords in its internal database. These are the user IDs and passwords you entered in the internal database when you created these user entries. For details, see "Setting Up Administrators" on page 190.

Figure 9.1 illustrates the authentication process.

Figure 9.1 CMS authentication of a user with *administrator* privileges



These are the steps shown in Figure 9.1:

1. An administrator opens Netscape Console and attempts to log in to the CMS window by entering the user ID and password at the login prompt. The server takes the administrator's user ID and password and binds them to privileged-user entries in its internal database.

2. If the user ID and password bind successfully to a user entry, authentication succeeds; otherwise, it fails.

   - If authentication fails, the server logs an error message and sends a rejection notification. See "Logs" on page 1043.

   - If authentication succeeds, the server checks the user's access rights (based on group memberships) to determine whether the user is authorized to perform the requested operation.

   If both authentication and authorization succeed, the server services the request. Otherwise, it rejects the request and logs the reason for the rejection.

Note Authentication for administrators is hardcoded; it is not configurable.

Authentication of Agents

When an agent makes a request to Certificate Management System (from the appropriate Agent Services interface), the server needs to authenticate the agent before processing the request. To facilitate this, Certificate Management System supports a certificate-based authentication method.

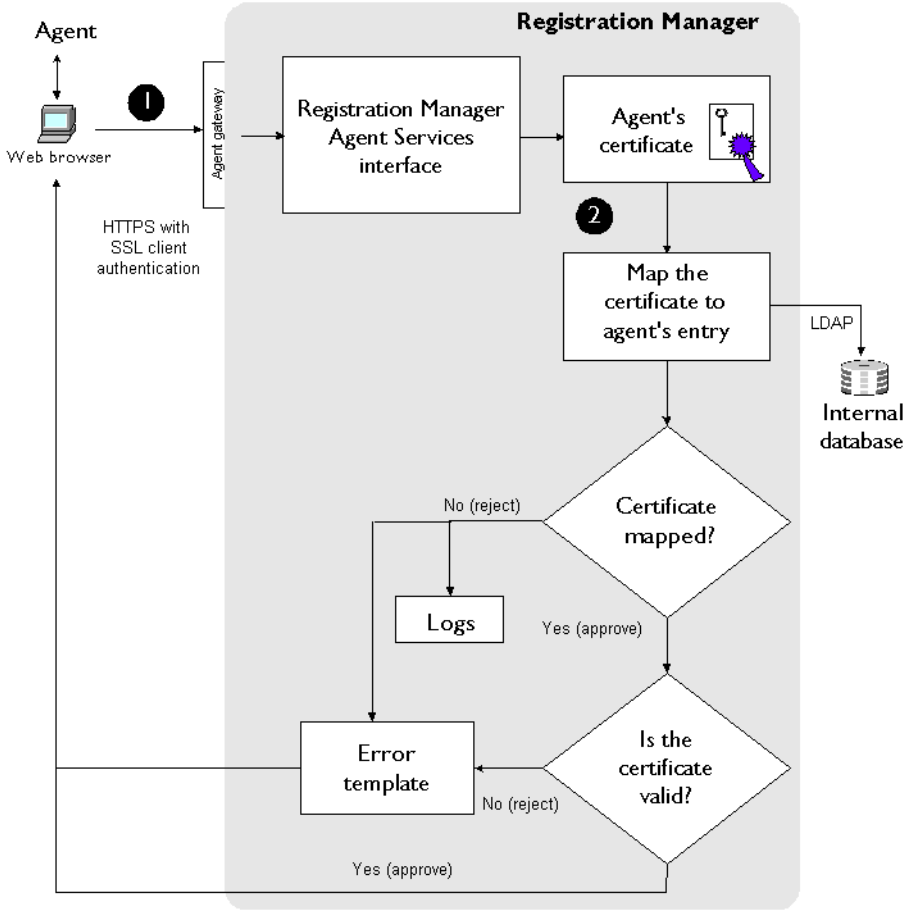
Certificate Management System identifies and authenticates a user with *agent* privileges by checking the user's SSL client certificate in its internal database. The certificates it checks are the ones you imported and stored in the internal database while creating or modifying the user entry. You create agent users for a CMS instance by adding their client certificates into the internal database and associating them with the corresponding users' identification information; for details, see "Setting Up Agents" on page 193.

When an agent makes a request to perform a privileged operation, the server requests SSL client authentication from the client that the agent has used to connect to the server. The server then uses the successfully SSL client-authenticated certificate to map to internal user entries for further checks. The server checks the certificate's subject name and issuer name against the list of privileged-user certificates stored in its internal database. If the certificate belongs to a privileged user who is authorized (based on group membership) to perform agent operations, the server allows the user to perform the requested operation. Otherwise, the server rejects the request and logs an appropriate message (for details, see "Logs" on page 1043).

Note Authentication for agents is hardcoded; it is not configurable.

Figure 9.2 shows how a Registration Manager authenticates and authorizes a Registration Manager agent.

Figure 9.2 Registration Manager authentication of a user with Registration Manager agent privileges



This example shows these steps:

1. An agent opens a web browser and enters the URL to the Registration Manager Agent Services interface hosted by the Registration Manager. The server requests the client for SSL client authentication. The client in turn

prompts the agent to specify the certificate that it should present to the server for authentication. The successfully SSL client authenticated certificate is presented to the Registration Manager.

2. Upon receiving the certificate, the Registration Manager performs the following authentication and authorization process:

   - First, it verifies that the certificate exists in its internal database. Next, it verifies that the certificate is a valid client certificate. If the certificate is valid, the Registration Manager proceeds. Otherwise (for example, if the certificate has expired or been revoked or was signed by an *untrusted* authority), the Registration Manager rejects the request, sends an error message to the agent, and logs a reason for the rejection.

     Note that the Registration Manager verifies the revocation status of the agent certificate if it has been issued by the Certificate Manager to which the Registration Manager is connected to; the Certificate Manager keeps a record of all the certificates it has issued and their current status in its internal database. However, if the agent certificate is issued by any other CA, the Registration Manager cannot verify the revocation status of the certificate; it can only verify that the certificate is valid and that it has been issued by a CA that the Registration Manager trusts. For details on configuring the Certificate Manager or Registration Manager to check the revocation status of its agents' certificates, see "Revocation Status Checking of Agent Certificates" on page 179.

     If the internal database contains an invalid certificate for an agent, the server rejects all requests from that agent. For the server to accept requests from that agent, you would have to replace the agent's invalid certificate in the internal database with a valid one. For details on how to do this, see "Changing a Privileged User's Certificate" on page 220.

   - The Registration Manager reads the user's subject name (in DN form) and the issuer name from the certificate. This combination is unique. It then finds the login name corresponding to this unique combination in its privileged-users list, which is stored in the internal database. If a login name is associated with the certificate, the Registration Manager proceeds. Otherwise, it rejects the request.

     The Registration Manager then checks the group memberships of the login name and the corresponding access rights to determine whether the user is authorized to perform the requested service.

If both authentication and authorization succeed, the Registration Manager services the request. Otherwise, it rejects the request and logs a reason for the rejection.

End-Entity Authentication

This section provides an overview of how Certificate Management System authenticates end entities during certificate enrollment, renewal, and revocation processes.

Authentication of End Entities During Certificate Enrollment

When an end entity submits a certificate request, a Certificate Manager or Registration Manager's first task is to identify and authenticate the end entity. The server must perform this task before it can register the end entity for certificate issuance. This task includes verifying the end entity's identity based on information the end entity provides and returning enough information about the end entity so that the subject name for the certificate can be constructed.

To cater to a variety of end-entity enrollment scenarios, Certificate Management System supports both manual and automated certificate issuance. For detailed description of authentication methods supported by the Certificate Manager and Registration Manager, see "Authentication Modules for End-Entity Enrollment" on page 319.

Authentication of End Users During Certificate Renewal

When an end user submits a certificate renewal request, the first step in the renewal process is for the Certificate Manager or Registration Manager to identify and authenticate the end user. This step includes making sure that the end user's current certificate is either "valid" or "expired" ("revoked" is not acceptable).

Certificate Management System verifies the authenticity of a certificate renewal request by mapping the subject name in the certificate being presented for renewal to certificates in its internal database. The server renews the certificate only if the subject name maps successfully to a certificate in its internal database. If the internal database contains more than one certificate with matching subject name as that the one presented by the end entity for client authentication, the server lists all the matching certificates and expects the end entity to pick one for renewal.

Here are a few things to keep in mind about certificate renewal:

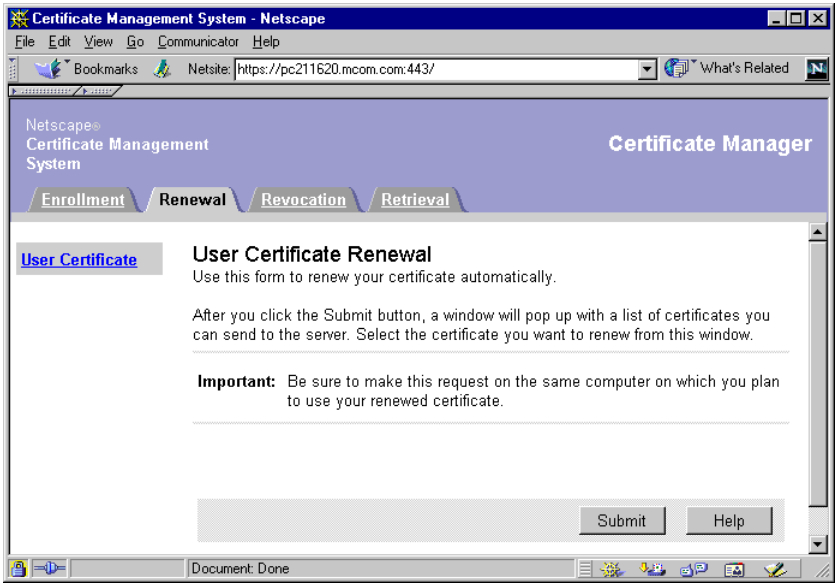
- The certificate being presented by the end user for renewal must be issued by a Certificate Manager.
- If the renewal request is processed by a Registration Manager, the end-user certificate presented must be issued by a Certificate Manager that the Registration Manager knows and is connected to; the Registration Manager forwards certificate requests to this Certificate Manager for signing.
- The certificate being presented by the end user for renewal must be currently valid or must have expired; it cannot have been revoked.
- The validity period of a renewed certificate is determined by the policy rule explained in "Renewal Validity Constraints Policy" on page 525. If the renewal lead time does not permit renewing, the server rejects the renewal request. Also, if the policy is disabled, renewal of certificates fails.
- If the certificate being presented by the end user has already been renewed, the server displays the URL for downloading the certificate.

  This situation may occur if the end user forgets to download the renewed certificate. It can also happen if the end user maintains two identical certificate databases on two machines, renews the certificate from one machine, and then tries to renew the same certificate from the other machine.

Certificate Renewal Form

The End Entity Services interface of the Certificate Manager and Registration Manager includes a default HTML form for renewing end users' certificates. The form is accessible from the Renewal tab as shown in Figure 9.3.

Figure 9.3 Certificate renewal form for end users



The default renewal form is preconfigured for SSL client authentication, enabling end users to renew their personal or client certificates by presenting valid or expired certificates.

If you want to change the form content to suit your organization's requirements, edit the following file:

```
<server_root>/cert-<instance_id>/web/ee/UserRenewal.html
```

For details on individual form elements, see the online help available by clicking the Help button on the form. For more information on customizing the form, see "Agent and End-Entity Interfaces" on page 893.

Authentication of End Users During Certificate Revocation

Certificates can be revoked by administrators, agents, and end users. When an end user submits a certificate revocation request, the first step in the revocation process is for the Certificate Manager or Registration Manager to identify and

authenticate the end user. The reason for this is when an end user attempts to revoke a certificate, the server needs to verify that the user is attempting to revoke his or her own certificate, not a certificate belonging to someone else.

Both Certificate Manager and Registration Manager support the following methods of revocation:

- SSL client authenticated revocation

  This method requires an end user to present a valid or revoked certificate that has the same subject name as the one he or she wants to revoke. Without the certificate, the user won't be able to revoke the certificate.

- Challenge-password-based revocation

  This method requires an end user to enroll for a personal certificate using the *manual enrollment* method. The reason for this is, by default, only the manual enrollment form includes fields for entering the challenge password when requesting a certificate; see "Enrollment Forms" on page 361. None of the other enrollment forms, for example directory-based or NIS server-based forms, by default allow end users to specify a challenge password.

  You can use the manual-enrollment form (`ManUserEnroll.html`) as a model and introduce the input fields for entering the challenge password in any of the other end user enrollment forms. Keep in mind that this feature is available for end-user certificates only; the feature is not available for other types of certificates.

  Revoking a certificate using the challenge password is useful in certain situations. For example, if you issue a single certificate to a user and the user is unable to use the certificate due to loss of corresponding key pair, it's not possible for the user to revoke his or her own certificate using the SSL client authenticated revocation method. If the user has a challenge password for the certificate, he or she can use it to revoke the certificate the server maintains in its database.

Forms for both methods are available through the End Entity Services interface (HTTPS only) of the Certificate Manager and Registration Manager; see "Certificate Revocation Forms" on page 317.

Here are a few common points to keep in mind about the automated revocation of end users' certificates:

- A Certificate Manager can revoke only those certificates that it has issued; it cannot revoke certificates issued by other CAs.

- If the revocation request is processed by a Registration Manager, it must be connected as a *trusted manager* to the Certificate Manager that has issued the certificate the user is attempting to revoke; the Registration Manager forwards certificate revocation requests to this Certificate Manager. For information on trusted managers, see "Trusted Managers" on page 181.
- The certificate the user attempts to revoke must be currently valid or must have expired; it cannot have been already revoked.
- At the time of revocation, the user can also specify additional details, such as the date of revocation and revocation reason, for each certificate or for the list as a whole.

SSL Client Authenticated Revocation

In an SSL client authenticated revocation method, the server expects the end user to present a certificate that has the same subject name as the one he or she wants to revoke and uses that for authentication purposes. The server verifies the authenticity of a revocation request by mapping the subject name in the certificate being presented for client authentication to certificates in its internal database. The server revokes the certificate only if the certificate maps successfully to one or more valid or expired certificates in its internal database.

After successful authentication, if the server detects only one valid or expired certificate with matching subject name as that of the one presented for client authentication, it revokes the certificate. If the server detects more than one valid or expired certificate with matching subject name, it lists all those certificates. The user can then either select the certificate to be revoked or revoke all certificates in the list.

Here are a few things, in addition to the ones listed on page 315, to keep in mind about SSL client authenticated revocation:

- The certificate being presented by the user for revocation must be issued by a Certificate Manager.
- If the revocation request is processed by a Registration Manager, the certificate presented for SSL client authentication must be issued by a Certificate Manager that the Registration Manager knows about and is connect to (the Registration Manager forwards certificate requests to this Certificate Manager for signing).

- The certificate being presented by the user for revocation must be currently valid or must have expired; it cannot have been already revoked.
- The user can revoke only certificates that contain the same subject name as the one in the certificate presented for authentication.

Challenge-Password-Based Revocation

A challenge password is a unique, alphanumeric string that the end user specifies when requesting a certificate; the user is expected to keep this password confidential and use it to authenticate to the server when revoking the certificate. When the server issues the certificate, it associates the password with the certificate, stores both the certificate and password in its internal database, and uses them later for authenticating any revocation requests.

In the challenge-password-based revocation method, the server expects the end user to specify the serial number of the certificate the user wants to revoke and the challenge password associated with the certificate. The server verifies the authenticity of a revocation request by mapping the serial number to the list of certificates in its internal database followed by mapping the challenge password specified to the one associated with the matching certificate it detects in the internal database.

The server revokes the certificate only if the certificate maps successfully to one or more valid or expired certificates in its internal database. If the server detects only one valid or expired certificate with a matching serial number and challenge password, it automatically revokes the certificate. If the server detects more than one valid or expired certificates with matching serial numbers, it lists all those certificates. The user can then select the certificate to be revoked or revoke all certificates in the list.

Here are a few things, in addition to the ones listed on page 315, to keep in mind about the challenge-password-based revocation:

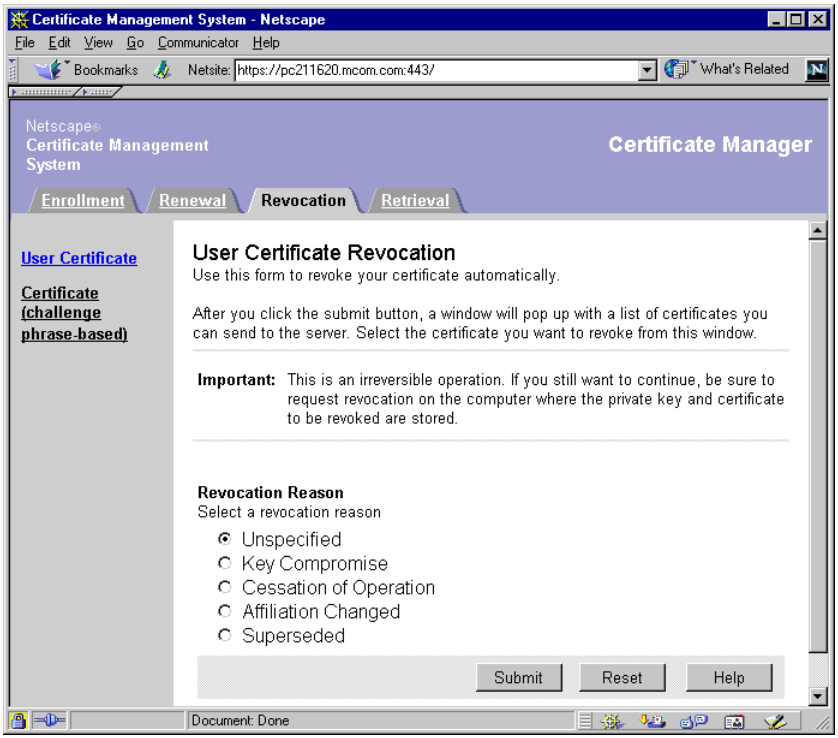
- The certificate being presented by the user for revocation must be issued by a Certificate Manager.
- The user must have requested the certificate using the *manual enrollment method*—only the default manual enrollment form includes fields for entering the challenge password when requesting a certificate; see "Enrollment Forms" on page 361.

- The user can revoke only those certificates that contain the specified serial number with the corresponding challenge password. For example, if there is a mismatch between the challenge password and serial number, the server rejects the revocation request.

Certificate Revocation Forms

The End Entity Services interface of the Certificate Manager and Registration Manager includes default HTML forms for both the SSL client authenticated revocation and challenge-password-based revocation. The forms are accessible from the Revocation tab. Figure 9.4 shows the form that enables end users to revoke their certificates using a challenge password. You can view the form that enables SSL client authenticated revocation by clicking the User Certificate link.

Figure 9.4 Form for SSL client authenticated certificate revocation



If you want to change the forms to suit your organization's requirements, you can edit the following files:

- `ChallengeRevoke1.html` (the form that allows challenge password based revocation of client or personal certificates)
- `UserRevocation.html` (the form that allows SSL client authenticated revocation of client or personal certificates)

Both the files are located here:

`<server_root>/cert-<instance_id>/web/ee`

For details on individual form elements, see the online help available by clicking the Help button on the form. For more information on customizing the form, see "Agent and End-Entity Interfaces" on page 893.

Chapter

10

Authentication Modules for End-Entity Enrollment

Certificate Management System provides a set of authentication modules that enable you to configure the server to authenticate end entities, based on specified criteria, when they enroll for a certificate. This chapter explains the authentication modules that are installed with the server—it lists and briefly describes the modules and then explains each one in detail. Before reading this chapter, you should have read the chapter "Introduction to Authentication" on page 305.

The chapter has the following sections:

Overview of Authentication Modules

Certificate Management System supports both manual and automated certificate issuance.

In the manual method of certificate issuance, end entities supply most of the information required by the server to formulate certificate requests and issue certificates. Manual issuance is also dependent on human agents; it requires that all end-entity certificate requests be approved by agents before the server can process the requests. To understand the role of an agent in your PKI, see "Agents" on page 173.

- In the automated method of certificate issuance, repositories, such as directories, supply part of the end-entity information. End entities only supply certain information—for example, a user ID and password—contained in the repository during certificate enrollment. The server uses this information for authenticating end entities before retrieving information required to formulate the certificate request from the repository.

For details on the manual method of certificate issuance, see "Manual Authentication" on page 323. For the automated method of certificate issuance, Certificate Management System provides a set of plug-in modules. Plug-in modules are implemented as Java classes and are registered in the CMS authentication framework. The Authentication Plugin Registration tab of the CMS window (see Figure 10.1) lists all the modules and the corresponding classes that are currently registered with a CMS instance.

Figure 10.1 Default authentication modules for end-user enrollment

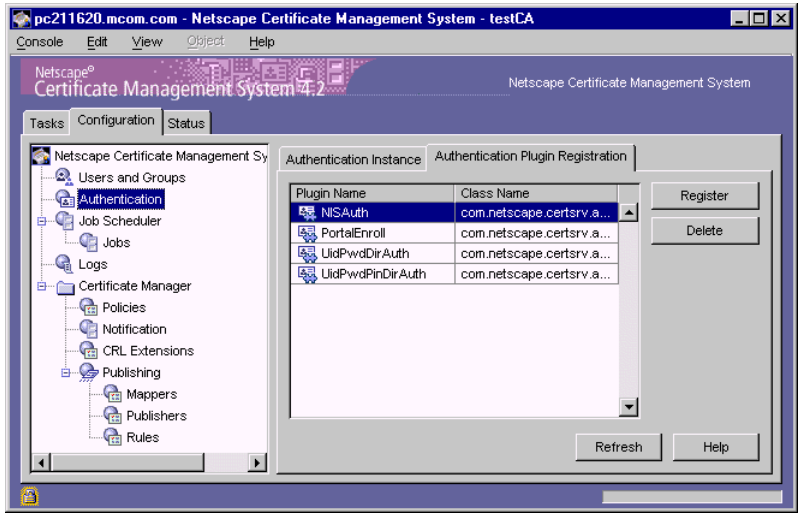


Table 10.1 lists the authentication modules provided for the Certificate Manager and Registration Manager; no authentication modules are provided for the Data Recovery Manager as it does not function as an enrollment authority in a PKI. You can use these modules to configure a Certificate Manager and Registration Manager to employ a specific authentication method during certificate enrollments.

Table 10.1 Authentication plug-in modules for end user certificate enrollments

Plug-in module name	Function
`NISAuth`	Authenticates end users based on their user IDs and passwords stored in a NIS server. Optionally, uses an LDAP directory for formulating certificate subject names. For details, see “NIS Server-Based Authentication” on page 340.
`PortalEnroll`	Authenticates online service users based on their user IDs and passwords stored in an LDAP directory. Also registers new users for the online service. For details, see “Portal Enrollment” on page 348.
`UidPwdDirAuth`	Authenticates end users based on their user IDs and passwords stored in an LDAP directory. For details, see “Manual Authentication” on page 323.
`UidPwdPinDirAuth`	Authenticates end users based on their user IDs, passwords, and PINs stored in an LDAP directory. For details, see “Directory-Based Authentication” on page 325.

Because large corporations typically store corporatewide user, group, and network-resource data in LDAP-compliant directories, the default authentication modules provided for automated certificate enrollment use an LDAP directory for authenticating users or for formulating certificate subject names, or for both. If you already have an LDAP-compliant directory, such as Netscape Directory Server, with end-user data, you can use that directory for any of the purposes mentioned above. For example, if you have an NIS server and LDAP directory installations, you can use the NIS server for authenticating end users and the directory for formulating certificate subject names; end users will be required to provide only their NIS user IDs and passwords during enrollment.

If you don't have a directory deployed, you may use the Netscape Directory Server instance created at the time of CMS installation; in the documentation, this instance is identified as the Configuration Directory. For a demonstration on how to use this directory for issuing certificates to end users, see Chapter 2: "Default Demo Installation" of the *Netscape Certificate Management System Installation and Deployment Guide*.

If you determine that the default authentication modules do not meet your requirements, you can develop a custom authentication module using the CMS SDK, which is available in the form of Java Docs at this location:

```
<server_root>/cms_sdk/sdkdocs
```

For general guidelines on developing custom authentication modules and adding them to the CMS authentication framework, see "Developing Custom Authentication Modules" on page 417. Be sure to take a look at the authentication-specific samples available at this location:

```
<server_root>/cms_sdk/samples/authentication
```

For instructions on how to configure a Certificate Manager and a Registration Manager to use one or more of the authentication methods, see "Setting Up Authentication for End-User Enrollment" on page 387.

Keep in mind that in an automated certificate management setup, the Certificate Manager and Registration Manager use the configured authentication methods only during certificate enrollment. During certificate renewal, the servers rely on end users SSL client certificate for automated renewal. For automated revocation, the users can use their SSL-client certificate or a challenge password. For more information, see sections "Authentication of End Users During Certificate Renewal" on page 312 and "Authentication of End Users During Certificate Revocation" on page 314.)

Certificate Management System also provides HTML forms-based interfaces for all the authentication methods it supports. Your end entities can use these forms for certificate enrollment. Explanation of each enrollment form, along with the corresponding authentication module, is covered in “Enrollment Forms” on page 361. Certificate renewal and revocation forms are covered as a part of those processes. For details on individual form elements in the enrollment, renewal, and revocation forms, see the online help available by clicking the Help buttons on the HTML forms. You can also customize these forms to suit to your organization’s requirements. For customization information, see “Agent and End-Entity Interfaces” on page 893.

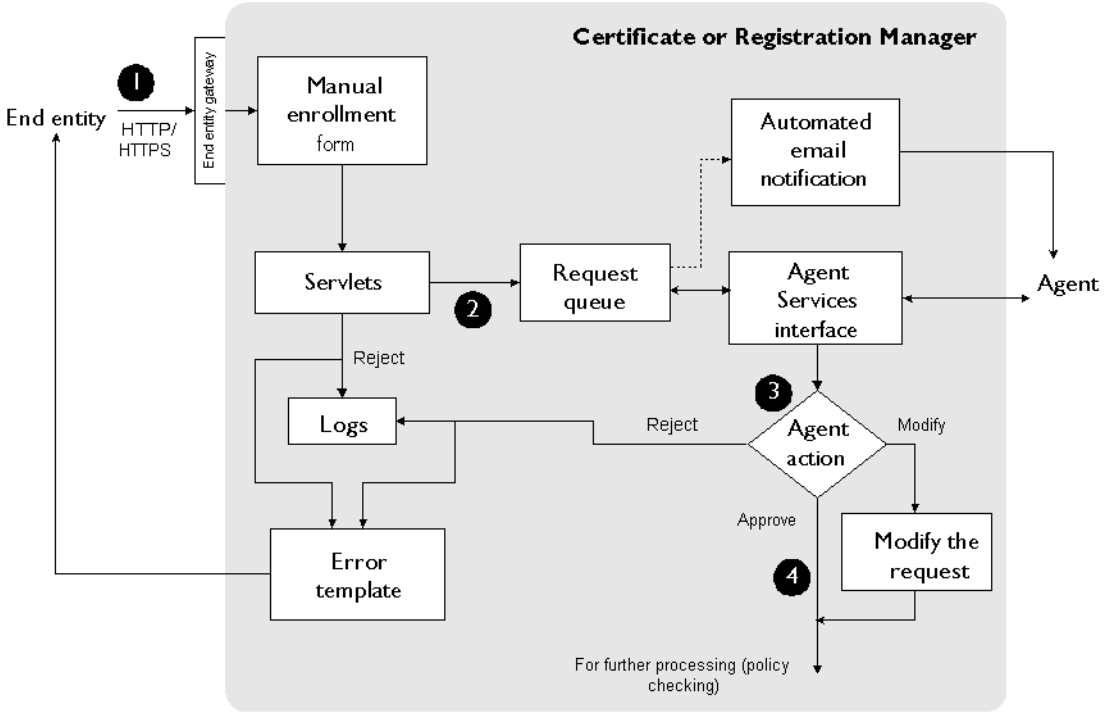
Manual Authentication

Manual authentication refers to operations which must be approved by a CMS agent, where no automated operation is possible. That is, a real person must log in to approve or reject request. By default, Certificate Management System provides manual-enrollment forms that enable you to request many types of certificates from the server. For details, see “Enrollment Forms” on page 361.

Note that the manual authentication method is hardcoded; you cannot configure it in any other way. This ensures that when the server receives requests that lack authentication credentials, it sends those requests to the request queue for agent approval. It also means that if you don’t configure a Certificate Manager or Registration Manager for any other authentication method, the server automatically sends all certificate-related requests to a queue where they await agent approval.

Figure 10.2 illustrates how the manual authentication method works during certificate enrollment.

Figure 10.2 Manual authentication of end entities during certificate enrollment



These are the steps shown in Figure 10.2:

1. In the manual enrollment form, the end entity enters the information needed to request a certificate and submits the request to the server.
2. When the server receives the request, it automatically lists the request in a *certificate request queue* for an agent to process.
3. An agent verifies the authenticity of the request.
   - If the request is from a valid end entity, the agent verifies that all the information the end entity has provided in the request is correct, makes required modifications, if any, and approves the certificate request for issuance.
   - If the request is from an invalid end entity, the agent rejects the request, which in turn triggers a rejection notification to the end entity.

4. When the server receives the agent-approved request, it subjects it to policy processing; for details, see "Policies" on page 485.

   - If the request fails any of the configured policies, the server rejects the request, logs an error message, and sends a rejection notification to the end entity.
   - If the request passes all the configured policies, the server issues the certificate.

   The certificate is delivered to the email address specified in the certificate request.

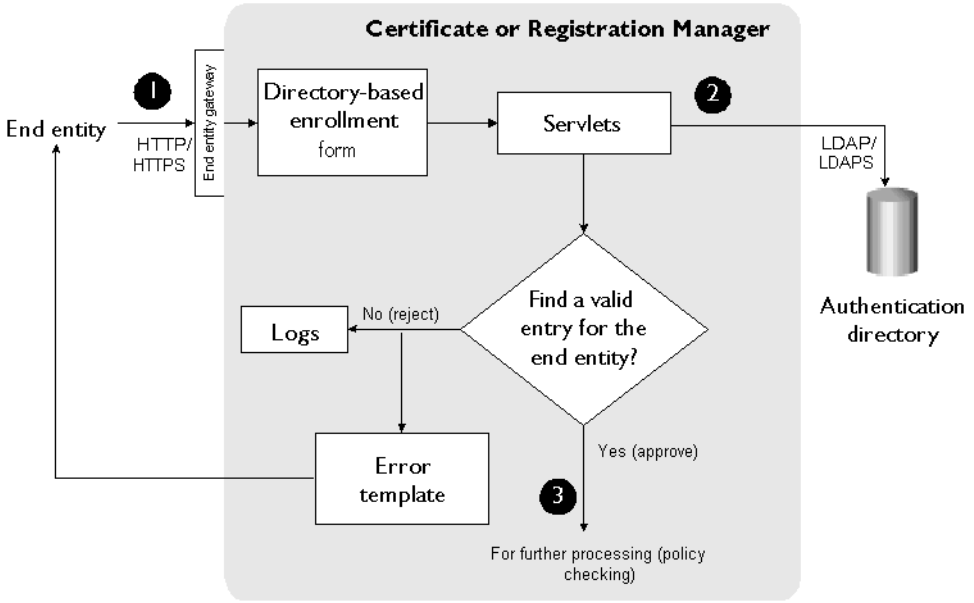
Directory-Based Authentication

The `UidPwdDirAuth` plug-in module implements the directory-based authentication method. You can use this module for authenticating end users during certificate enrollment. You can use this module for authenticating unprivileged users in the global LDAP domain.

As part of configuring a Certificate Manager and a Registration Manager, or both, for authentication, you specify an LDAP directory that the server must use to authenticate end users. End users enroll for a certificate by entering their user IDs and passwords for this authentication directory in an HTML form that is served by a Certificate Manager or Registration Manager (see "Enrollment Forms" on page 361). Once the server successfully authenticates an end user, it retrieves the rest of the information required to formulate the certificate from the directory.

Figure 10.3 illustrates how authentication based on a user ID and password works during certificate enrollment.

Figure 10.3 User ID- and password-based authentication of an end user



These are the steps shown in Figure 10.3:

1. In the directory-based certificate enrollment form, the end user enters a user ID and password for the directory and submits the request to a Certificate Manager or Registration Manager.

2. When the server receives the request, it looks up the directory that is configured for authenticating end users. The server verifies the authenticity of the user by checking the directory entries.

   - If the end user does not have a valid entry in the directory, the server rejects the request, logs an error message, and sends a rejection notification to the user.

   - If the end user has a valid entry in the directory, the server retrieves all the information required to construct the subject name for the user's certificate.

If, for some reason, the directory to which the server binds for authenticating the user ID and password is unavailable, the server returns an LDAP error code and writes it to the log. When troubleshooting any authentication failures, if you see a log entry with an LDAP error code, look up the corresponding error code at this URL:
`http://help.netscape.com/kb/server/970303-9.html`

A sample log entry with an LDAP error code is shown below:

```
28/Jun/1999:18:40:25 -0700] conn=0 op=7 RESULT err=32 tag=101 nentries=0 etime=0]
```

3. Next, the server subjects the certificate request to policy processing; for details, see "Policies" on page 485.

   - If the request fails any of the configured policies, the server rejects the request, logs an error message, and sends a rejection notification to the end entity.
   - If the request passes all the configured policies, the server issues the end user a certificate.

     The end user gets the certificate, which, if the server is configured to do so, is delivered to the email address specified in the request or in the directory; for information on configuring a Certificate Manager or Registration Manager to send automated notifications, see "Notifications of Certificate Issuance to End Entities" on page 450.

UidPwdDirAuth Module

The Java class that implements the directory-based authentication method is identified as follows:

`com.netscape.certsrv.authentication.UidPwdDirAuthentication`

- In the configuration file, the module is identified as follows:
  `auths.impl.UidPwdDirAuth.class=com.netscape.certsrv.authentication.UidPwdDirAuthentication`
- In the CMS window, the module is identified as follows: `UidPwdDirAuth`

Figure 10.4 shows how configurable parameters of the `UidPwdDirAuth` module are displayed in the CMS window.

Figure 10.4 Parameters defined in the UidPwdDirAuth module

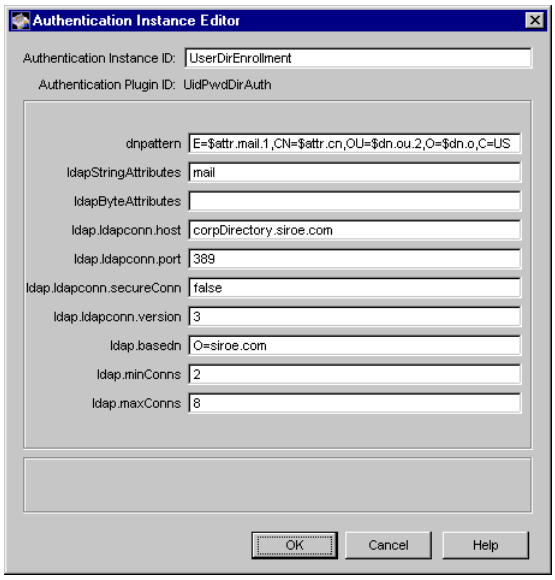


Table 10.2 gives details about each of these parameters and their values.

Table 10.2 Description of parameters defined in the UidPwdDirAuth module

Parameter	Description
`dnpattern`	Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN. Permissible values: Any valid DN string composed from standard DN attributes, which must be separated by commas; see "DNs in Certificate Management System" on page 1157. The syntax is illustrated in the following example: `E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US` This sample configuration specifies that the subject name should be formulated as follows: • `E` = the first `mail` LDAP attribute value in the user's entry • `CN` = the (first) `cn` LDAP attribute value in the user's entry • `OU` = the second `ou` value in the user's entry DN • `O` = the (first) `o` value in the user's entry DN • `C` = the string `US` If this parameter value is empty or not set, the server uses `E=$attr.mail, CN=$attr.cn, O=$dn.o, C=$dn.c` as the DN pattern. This default DN pattern works well with Netscape Communicator and other browsers. For Communicator, if you leave out `E=` in end-user certificates, S/MIME may not work correctly (assuming lack of other extensions in the certificate). Also, if `C=` and `O=` are left out, certificate display looks strange in Communicator (when the Display Certificate button is clicked).

Table 10.2 Description of parameters defined in the UidPwdDirAuth module (Continued)

Parameter	Description
`ldapStringAttributes`	Specifies the list of LDAP string attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token—that is, values retrieved from this parameter can be used by policy modules to formulate subject names for certificates or to make other policy decisions. For details, see "Subject Alternative Name Extension Policy" on page 668. Entering values for this parameter is optional. Permissible values: Any valid LDAP string attributes, separated by commas. Example: `mail` (This sample configuration specifies that the value of the `mail` attribute should be stored in the authentication token.)
`ldapByteAttributes`	Specifies the list of LDAP byte (binary) attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token for use by other modules—that is, values retrieved from this parameter can be used by policy modules to make certain policy decisions or to add additional information to users' certificates. For example, assume you have defined an LDAP binary attribute for storing users' pictures or fingerprints in your directory. You could develop a policy plug-in that adds users' pictures to their certificates as extensions. Entering values for this parameter is optional. Permissible values: Any valid LDAP byte attributes, separated by commas. Example: `jpegPhoto` This sample configuration specifies that the value of the LDAP attribute named `jpegPhoto` (which is included in the standard `inetOrgPerson` object class) should be stored in the authentication token and be used to put the user's picture in his or her certificate.

Table 10.2 Description of parameters defined in the UidPwdDirAuth module (Continued)

Parameter	Description
`ldap.ldapconn.host`	Specifies the host name of the authentication directory. Permissible values: The name must be in the `<machine_name>.<your_domain>.<domain>` form. Example: `corpDirectory.siroe.com`
`ldap.ldapconn.port`	Specifies the TCP/IP port at which the authentication directory listens to requests from Certificate Management System. Permissible values: Any valid port number. Example: `389`
`ldap.ldapconn.secureConn`	Specifies the type—SSL or non-SSL—of the port at which the authentication directory listens to requests from Certificate Management System. • Check the box if the port is an SSL (HTTPS) port. If your authentication directory is configured for SSL-enabled communication (with or without SSL client authentication), choose this option. • Leave the box unchecked if the port is a non-SSL (HTTP) port. If your authentication directory is configured for basic authentication, choose this option (default).
`ldap.ldapconn.version`	Specifies the LDAP protocol version. Permissible values: `2` or `3`. • `2` specifies LDAP version 2. If your authentication directory is based on Netscape Directory Server 1.x, choose `2`. • 3 specifies LDAP version 3. For Directory Server versions 3.x and later, choose `3`. Example: `3`
`ldap.basedn`	Specifies the base DN for searching the authentication directory—the server uses the value of the `uid` field from the HTTP input (what a user enters in the enrollment form) and the base DN to construct an LDAP search filter. Permissible values: Any valid DN string of up to 255 characters. Example: `O=siroe.com`

Table 10.2 Description of parameters defined in the UidPwdDirAuth module (Continued)

Parameter	Description
`ldap.minConns`	Specifies the minimum number of connections permitted to the authentication directory. Permissible values: `1` to `3`. Example: `2`
`ldap.maxConns`	Specifies the maximum number of connections permitted to the authentication directory. Permissible values: `3` to `10`. Example: `8`

Directory- and PIN-Based Authentication

The `UidPwdPinDirAuth` plug-in module implements the directory- and PIN-based authentication method. You can use this module for authenticating users in the global LDAP domain during certificate enrollment. This authentication method is functionally very similar to the directory-based authentication explained in "Directory-Based Authentication" on page 325, except that for stronger authentication you combine a PIN or one-time password with the end users' user IDs and passwords.

As a part of setting up a Certificate Manager or a Registration Manager, or both for end-user authentication, you specify the LDAP directory that the server must use to authenticate end users. End users enroll for a certificate by entering their user IDs, passwords, and PINs in an HTML form that is served by the Certificate Manager or Registration Manager (see "Enrollment Forms" on page 361). Once the server successfully authenticates an end user, it retrieves the rest of the information required to formulate the certificate from the directory. You can also configure the server to either retain or remove the PIN from the directory following successful authentication.

Normally, user entries in directories do not contain PINs. In order to use the `UidPwdPinDirAuth` module, you must first populate the directory that you intend to use for authentication with unique PINs for users; each user to whom you intend to issue a certificate must know his or her PIN at the time of

certificate enrollment, as he or she will be required to enter it in the enrollment form. To aid you in the process of generating unique PINs for users and adding them to the directory, Certificate Management System provides a command-line tool called the PIN Generator. The section "Using the PIN Generator Tool" on page 369 explains how to use this tool in detail.

UidPwdPinDirAuth Module

The Java class that implements the directory- and PIN-based authentication method is identified as follows:

```
com.netscape.certsrv.authentication.UidPwdPinDirAuthentication
```

- In the configuration file, the module is identified as follows:
  `auths.impl.UidPwdPinDirAuth.class=com.netscape.certsrv.authentication.UidPwdPinDirAuthentication`
- In the CMS window, the module is identified as follows:
  `UidPwdPinDirAuth`

Figure 10.5 shows how configurable parameters of the `UidPwdPinDirAuth` module are displayed in the CMS window.

Figure 10.5 Parameters defined in the UidPwdPinDirAuth module

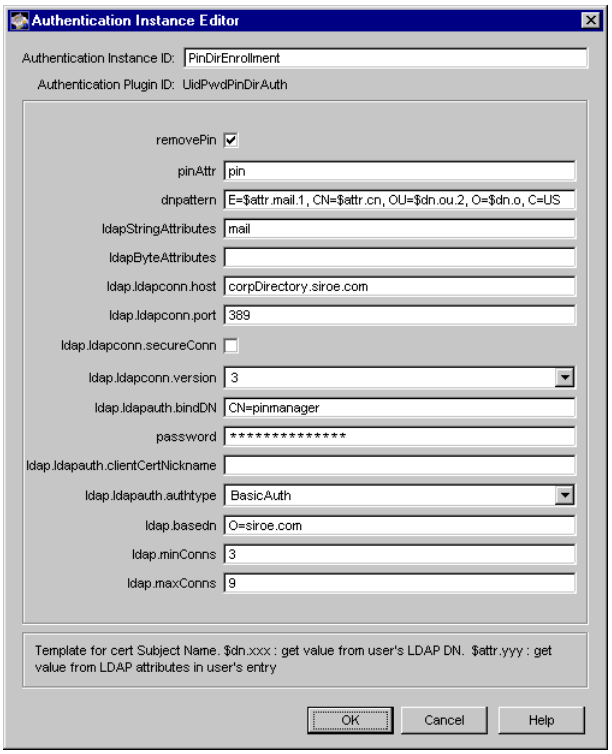


Table 10.3 gives details about each of these parameters.

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module

Parameter	Description
`removePin`	Specifies whether to remove PINs from the authentication directory (after end users successfully authenticate). Removing PINs from the directory restricts users from enrolling more than once, and thus prevents them from getting more than one certificate. • Check the box if you want the server to remove PINs from the directory after successful authentication. If you set the value to `true`, you must also specify the values for the `ldap.ldapauth.bindDN` and `password` parameters. • Uncheck the box if you want the server to leave PINs in the directory after authentication.

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module (Continued)

Parameter	Description
`pinAttr`	Specifies the authentication directory attribute for PINs. If you used the *PIN Generator* utility (provided with Certificate Management System), the attribute is specified by the value of the `objectclass` parameter; the default value for this parameter is `pin`. For details, see "Arguments" on page 370. Permissible values: Any valid attribute name. Example: `pin`
`dnpattern`	Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN. Permissible values: Any valid DN string composed from standard DN attributes, which must be separated by commas; see "DNs in Certificate Management System" on page 1157. The syntax is illustrated in the following example: `E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US` This sample configuration specifies that the subject name should be formulated as follows: • `E` = the first `mail` LDAP attribute value in the user's entry • `CN` = the (first) `cn` LDAP attribute value in the user's entry • `OU` = the second `ou` value in the user's entry DN • `O` = the (first) `o` value in the user's entry DN • `C` = the string `US` If this parameter value is empty or not set, the server uses `E=$attr.mail, CN=$attr.cn, O=$dn.o, C=$dn.c` as the DN pattern. This default DN pattern works well with Netscape Communicator and other browsers. For Communicator, if you leave out `E=` in end-user certificates, S/MIME may not work correctly (assuming lack of other extensions in the certificate). Also, if `C=` and `O=` are left out, certificate display looks strange in Communicator (when the Display Certificate button is clicked).

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module (Continued)

Parameter	Description
`ldapStringAttributes`	Specifies the list of LDAP string attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token—that is, values retrieved from this parameter can be used by policy modules to formulate subject names for certificates or to make other policy decisions. For details, see "Subject Alternative Name Extension Policy" on page 668. Entering values for this parameter is optional. Permissible values: Any valid LDAP string attributes, separated by commas. Example: `mail` (This sample configuration specifies that the value of the `mail` attribute should be stored in the authentication token.)
`ldapByteAttributes`	Specifies the list of LDAP byte (binary) attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token for use by other modules—that is, values retrieved from this parameter can be used by policy modules to make certain policy decisions or to add additional information to users' certificates. For example, assume you have defined an LDAP binary attribute for storing users' pictures or fingerprints in your directory. You could develop a policy plug-in that adds users' pictures to their certificates as extensions. Entering values for this parameter is optional. Permissible values: Any valid LDAP byte attributes, separated by commas. Example: `jpegPhoto` This sample configuration specifies that the value of the LDAP attribute named `jpegPhoto` (which is included in the standard `inetOrgPerson` object class) should be stored in the authentication token and be used to put the user's picture in his or her certificate.

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module (Continued)

Parameter	Description
`ldap.ldapconn.host`	Specifies the host name of the authentication directory. Permissible values: The name must be in the `<machine_name>.<your_domain>.<domain>` form. Example: `corpDirectory.siroe.com`
`ldap.ldapconn.port`	Specifies the TCP/IP port at which the authentication directory listens to requests from Certificate Management System. Permissible values: Any valid port number. Example: `389`
`ldap.ldapconn.secureConn`	Specifies the type—SSL or non-SSL—of the port at which the authentication directory listens to requests from Certificate Management System. • Check the box if the port is an SSL (HTTPS) port. If your authentication directory is configured for SSL-enabled communication (with or without SSL client authentication), choose this option. • Leave the box unchecked if the port is a non-SSL (HTTP) port. If your authentication directory is configured for basic authentication, choose this option (default).
`ldap.ldapconn.version`	Specifies the LDAP protocol version. Permissible values: `2` or `3`. • `2` specifies LDAP version 2. If your authentication directory is based on Netscape Directory Server 1.x, choose `2`. • `3` specifies LDAP version 3. For Directory Server versions 3.x and later, choose `3` (default). Example: `3`

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module (Continued)

Parameter	Description
`ldap.ldapauth.bindDN`	Specifies the user entry to bind as when removing PINs from the authentication directory. You need to specify this parameter only if you've selected `removePin`. It is recommended that you create and use a separate user entry that has permission to modify only the PIN attribute in the directory. For example, don't use the directory manager's entry as it has privileges to modify the entire directory content. Permissible values: A valid bind DN. Example: `CN=pinmanager`
`password`	Specifies the password associated with the DN specified by the `ldap.ldapauthbindDN` parameter. when you save your changes, the server stores the password in the single sign-on password cache and uses it for subsequent start ups (see "Required Start-up Information" on page 128). You need to specify this parameter only if you've selected `removePin`.
`ldap.ldapauth.` `clientCertNickname`	Specifies the nickname or the friendly name of the certificate to be used for SSL client authentication to the authentication directory in order to remove PINs. Make sure that the certificate is valid and has been signed by a CA that is trusted in the authentication directory's certificate database, and that the authentication directory's `certmap.conf` file has been configured to correctly map the certificate to a DN in the directory. (This is needed for PIN removal only.) Permissible values: Enter the name of a currently valid CMS certificate, for example, its SSL server certificate. Example: `Server-Cert`

Table 10.3 Description of parameters defined in the UidPwdPinDirAuth module (Continued)

Parameter	Description
`ldap.ldapauth.authtype`	Specifies the authentication type—basic authentication or SSL client authentication—required in order to remove PINs from the authentication directory. Permissible values: `BasicAuth` or `SslClientAuth`. • `BasicAuth` specifies basic authentication. If you choose this option, be sure to enter the correct values for `ldap.ldapauth.bindDN` and `password` parameters; the server uses the DN from the `ldap.ldapauth.bindDN` attribute to bind to the directory (default). • `SslClientAuth` specifies SSL client authentication. If you choose this option, be sure to set the value of the `ldap.ldapconn.secureConn` parameter to `true` and the value of the `ldap.ldapauth.clientCertNickname` parameter to the nickname of the certificate to be used for SSL client authentication. Example: `BasicAuth`
`ldap.basedn`	Specifies the base DN for searching the authentication directory—the server uses the value of the `uid` field from the HTTP input (what a user enters in the enrollment from) and the base DN to construct an LDAP search filter. Permissible values: Any valid DN string of up to 255 characters. Example: `O=siroe.com`
`ldap.minConns`	Specifies the minimum number of connections permitted to the authentication directory. Permissible values: `1` to `3`. Example: `3`
`ldap.maxConns`	Specifies the maximum number of connections permitted to the authentication directory. Permissible values: `3` to `10`. Example: `9`

NIS Server-Based Authentication

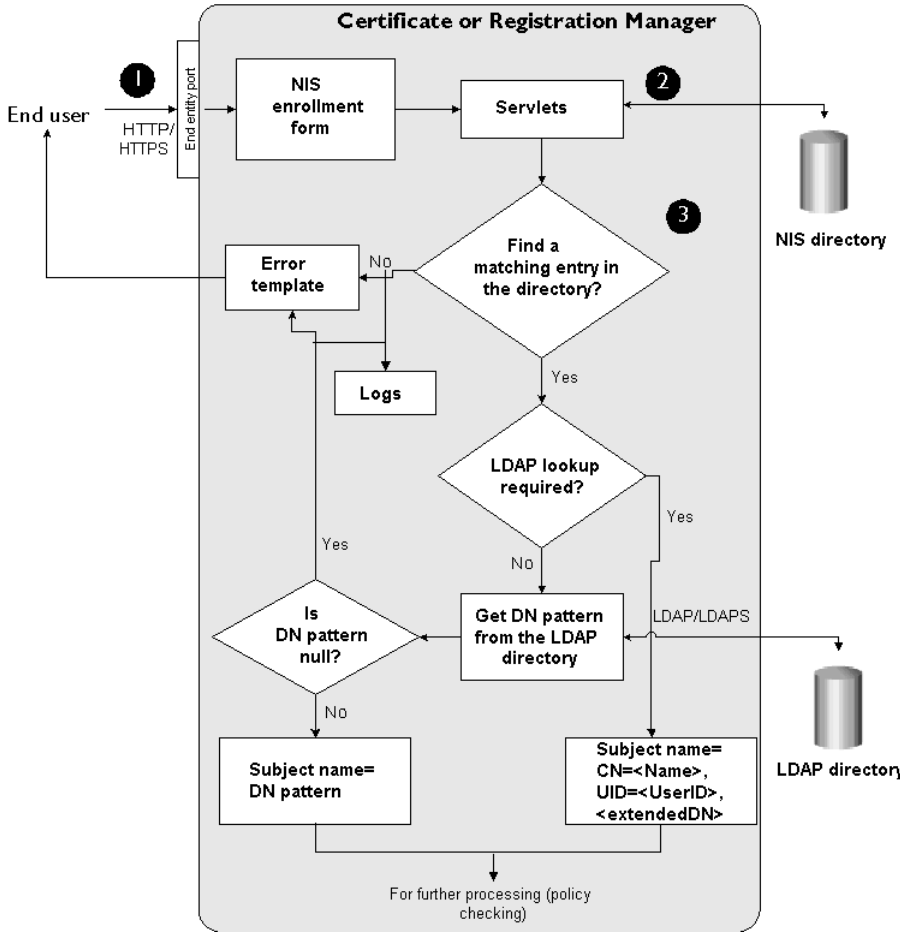
The `NISAuth` module implements the NIS server-based authentication. You can use the module for authenticating unprivileged users in the NIS domain during certificate enrollment. The module enables you to deploy Public Key Infrastructure (PKI) leveraging an existing NIS server installation—it enables you to configure a Certificate Manager or Registration Manager to authenticate end users, based on their user IDs and passwords stored in an existing NIS server, and to issue certificates.

Optionally, you can configure the authentication module to do an *LDAP correlation*—that is, use the NIS directory to authenticate users based on the user ID and password they enter in the enrollment form, but compose certificate subject names from an LDAP-compliant directory, such as Netscape Directory Server. When using an LDAP directory to compose subject names, you can configure the module to search for and retrieve specific LDAP attribute values from the directory. The ability of the module to use an LDAP directory to form certificate subject names is useful in cases where the NIS server only stores user IDs and passwords and you don't want to formulate subject names using just common names and user IDs.

In the absence of an LDAP directory, subject names of all certificates issued by the server will be of the form `CN=<FirstName LastName>,UID=<UserID>`, where `First Name` and `Last Name` is a user's first and last names as specified in the NIS directory, and `UserID` is the user's NIS ID. To accommodate scenarios where the default subject-name form isn't adequate, the module supports a parameter named `extendedDN`. This parameter enables you to specify a suffix that the server should use for extending the default subject DN pattern.

Figure 10.6 illustrates how the NIS authentication module works during certificate enrollment.

Figure 10.6 NIS server-based authentication of an end user



These are the steps shown in Figure 10.6:

1. In the NIS server-based certificate enrollment form, the end user enters his or her user ID and password for the NIS server and submits the request to a Certificate Manager or Registration Manager.

2. When the server receives the request, it looks up the NIS server that is configured for authenticating end users. The server verifies the authenticity of the end user by checking the entries.

- If the end user does not have a valid entry in the NIS server, the Certificate Manager or Registration Manager rejects the request, logs an error message, and sends a rejection notification to the user.

- If the end user has a valid entry in the NIS server, the Certificate Manager or Registration Manager checks to see if any LDAP directory has been configured for retrieving attributes for constructing the certificate subject name. If a directory is specified, the server checks it for the user's entry, retrieves all the information required to construct the subject name, and adds the subject name to the certificate request. If a directory is unspecified, the server uses the NIS user's name, user ID, and extended DN (if specified) for the subject name.

If, for some reason, the directory to which the server binds for retrieving user attributes is unavailable, the server writes the appropriate LDAP error code to the log. When checking the log for troubleshooting any authentication failures, if you see a log entry with an LDAP error code, look up the corresponding error code at this URL:

`http://help.netscape.com/kb/server/970303-9.html`

A sample log entry with an LDAP error code is shown below:

```
30/Dec/1999:18:40:25 -0700] conn=0 op=7 RESULT err=32 tag=101 nentries=0 etime=0]
```

3. Next, the server subjects the certificate request to policy processing; for details, see "Policies" on page 485.

   - If the request fails any of the configured policies, the server rejects the request, logs an error message, and sends a rejection notification to the end user.

   - If the request passes all the configured policies, the server issues the end user a certificate.

     The end user gets the certificate, which, if the server is configured to do so, is delivered to the email address specified in the request or in the directory; for information on configuring a Certificate Manager or Registration Manager to send automated notifications, see "Notifications of Certificate Issuance to End Entities" on page 450.

NISAuth Module

The Java class that implements the NIS server-based authentication module is identified as follows:

`com.netscape.certsrv.authentication.NISAuth`

- In the configuration file, the module is identified as follows: `auths.impl.NISAuth.class=com.netscape.certsrv.authentication.NISAuth`
- In the CMS window, the module is identified as follows: `NISAuth`

Figure 10.7 shows how configurable parameters of the `NISAuth` module are displayed in the CMS window.

Figure 10.7 Parameters defined in the NISAuth module

Table 10.4 gives details about each of these parameters and their values.

Table 10.4 Description of parameters defined in the NISAuth module

Parameter	Description
`nisserver`	Specifies the NIS server name. (In Unix, use the `ypwhich` command to find the NIS server name.) Permissible values: A valid server name. Example: `myServer`
`nisdomain`	Specifies the NIS domain name. (In Unix, use the `domainname` command to find the domain name.) Permissible values: A valid domain name. Example: `siroe.com`
`dnpattern`	Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN. Permissible values: Any valid DN string composed from standard DN attributes, which must be separated by commas; see "DNs in Certificate Management System" on page 1157. The syntax is illustrated in the following example: `E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US` This sample configuration specifies that the subject name should be formulated as follows: • `E` = the first `mail` LDAP attribute value in the user's entry • `CN` = the (first) `cn` LDAP attribute value in the user's entry • `OU` = the second `ou` value in the user's entry DN • `O` = the (first) `o` value in the user's entry DN • `C` = the string `US` If this parameter value is empty or not set, the server uses `E=$attr.mail, CN=$attr.cn, O=$dn.o, C=$dn.c` as the DN pattern. This default DN pattern works well with Netscape Communicator and other browsers. For Communicator, if you leave out `E=` in end-user certificates, S/MIME may not work correctly (assuming lack of other extensions in the certificate). Also, if `C=` and `O=` are left out, certificate display looks strange in Communicator (when the Display Certificate button is clicked).

Table 10.4 Description of parameters defined in the NISAuth module (Continued)

Parameter	Description
`extendedDN`	Specifies the suffix that the server should use for extending the default subject DN when an LDAP directory for retrieving such information is not specified. The value you specify in this field is used by the sever to suffix the default subject name in certificates, which is in the form `CN=<FirstName LastName>,UID=<UserID>`. Example: If you assign `OU=People,O=siroe.org,C=US` as the extended DN, subject names in certificates would be of this form: `CN=<FirstName LastName>,UID=<UserID>,OU=People, O=siroe.org,C=US`
`ldapStringAttributes`	Specifies the list of LDAP string attributes that should be considered *authentic* for the end entity. If specified, the values corresponding to these attributes will be copied from the authentication directory into the authentication token—that is, values retrieved from this parameter can be used by policy modules to formulate subject names for certificates or to make other policy decisions. For details, see "Subject Alternative Name Extension Policy" on page 668. Entering values for this parameter is optional. Permissible values: Any valid LDAP string attributes, separated by commas. Example: `mail` (This sample configuration specifies that the value of the `mail` attribute should be stored in the authentication token.)

Table 10.4 Description of parameters defined in the NISAuth module (Continued)

Parameter	Description
`ldapByteAttributes`	Specifies the list of LDAP byte (binary) attributes that should be considered *authentic* for the end user. If specified, the values corresponding to these attributes will be copied from the LDAP directory into the authentication token for use by other modules—that is, values retrieved from this parameter can be used by policy modules to make certain policy decisions or to add additional information to users' certificates. For example, assume you have defined an LDAP binary attribute for storing users' pictures or fingerprints in your directory. You could develop a policy plug-in that adds users' pictures to their certificates as extensions. Entering values for this parameter is optional. Permissible values: Any valid LDAP byte attributes, separated by commas. Example: `jpegPhoto` This sample configuration specifies that the value of the LDAP attribute named `jpegPhoto` (which is included in the standard `inetOrgPerson` object class) should be stored in the authentication token and be used to put the user's picture in his or her certificate.
`ldap.ldapconn.host`	Specifies the host name of the LDAP directory. Permissible values: The name must be in the `<machine_name>.<your_domain>.<domain>` form. Example: `corpDirectory.siroe.com`
`ldap.ldapconn.port`	Specifies the TCP/IP port at which the LDAP directory listens to requests from Certificate Management System. Permissible values: Any valid port number. Example: `389`

Table 10.4 Description of parameters defined in the NISAuth module (Continued)

Parameter	Description
`ldap.ldapconn.secureConn`	Specifies the type—SSL or non-SSL—of the port at which the LDAP directory listens to requests from Certificate Management System. Permissible values: `true` or `false`. • `true` specifies that the port is an SSL (HTTPS) port. If your directory is configured for SSL-enabled communication (with or without SSL client authentication), choose this option. • `false` specifies that the port is a non-SSL (HTTP) port. If your directory is configured for basic authentication, choose this option. Example: `false`
`ldap.ldapconn.version`	Specifies the LDAP protocol version of the LDAP directory. Permissible values: `true` or `false`. • `2` specifies LDAP version 2. If your directory is based on Netscape Directory Server 1.x, choose `2`. • 3 specifies LDAP version 3. For Directory Server versions 3.x and later, choose `3`. Example: `3`
`ldap.basedn`	Specifies the base DN for searching the LDAP directory—the server uses the value of the `uid` field from the HTTP input (what a user enters in the enrollment form) and the base DN to construct an LDAP search filter. Permissible values: Any valid DN string of up to 255 characters. Example: `O=siroe.com`
`ldap.minConns`	Specifies the minimum number of connections permitted to the LDAP directory. Permissible values: `1` to `3` Example: `2`

Table 10.4 Description of parameters defined in the NISAuth module (Continued)

Parameter	Description
`ldap.maxConns`	Specifies the maximum number of connections permitted to the LDAP directory. Permissible values: `3` to `10` Example: `10`

Portal Enrollment

The `PortalEnroll` module implements portal enrollment. This module enables you to issue certificates and create directory entries for users who do not yet have an entry in the directory. For example, if your company runs a portal service, such as Netscape Netcenter™, you can use the `PortalEnroll` module to issue certificates to new users when they register for the online service. You can also use the module to authenticate and issue certificates to your extranet users. For example, if you have deployed extranets for partners and vendors, you can use the module to authenticate and issue certificates to these users when they register for the service.

The `PortalEnroll` module does following:

- Performs dual operations, registration and authentication, eliminating the need for users to use separate forms to register for an online service and to request a certificate; the module enables deployment of certificates along with registration in an LDAP-compliant directory.
- Verifies the uniqueness of the new user's chosen user name against an LDAP-compliant user directory and uses the user name as the only authentication token required to obtain a certificate.
- Uses the information from the enrollment form to create new user entries and update directory entry attributes for unique usernames.
- Leverages an existing LDAP-compliant user directory, typically used for storing user information.

There are many advantages in issuing certificates to your user community:

- Certificates enable you to uniquely identify users and establish a relationship with users in that you can use their identities to track services and features utilized by these users and use this information to offer customized services to them—certificates become equivalent to the way online services utilize cookies for personalization.

- Certificates also enable you to make your online service subscription based—because a certificate's life is tied to its validity period, by issuing certificates with specific validity period you can enforce users to subscribe to your online service by renewing their certificate before its expiry.

- Certificates also enable you to remove people from your user base and add them back after giving them a credential—by making a certificate issued to a new user expire after a specific validity period you can restrict that user from using your service, and put the user back on service by forcing the user to renew the expired certificate after giving them a credential. For example, assume you have an extranet deployed for your partners. You have no prior knowledge of people who will register as your partners, but you want them to register and you want to trust the information they provide during registration. By issuing them a certificate with a short validity period you can limit them from using your service for that period. In the meantime, you can verify their registration data and decide whether to allow them to continue using your service; if you want them to be your partners, you allow them to renew their certificates before they expire; if you don't want them as your partners, you reject their certificate renewal requests.

  Note that Certificate Management System can send automated renewal notifications to users before their certificates expire; see "Certificate Renewal Notifications" on page 435.

Functionally, the portal authentication module is very similar to the directory-based authentication module (see "Directory-Based Authentication" on page 325) except that instead of binding to the directory as the enrolling user, Certificate Management System binds as some directory account with permission to create and update user entries. The server then queries the directory for the user name specified by the user and if it doesn't find a match, it adds the entry with all the standard LDAP field names that match the directory attributes.

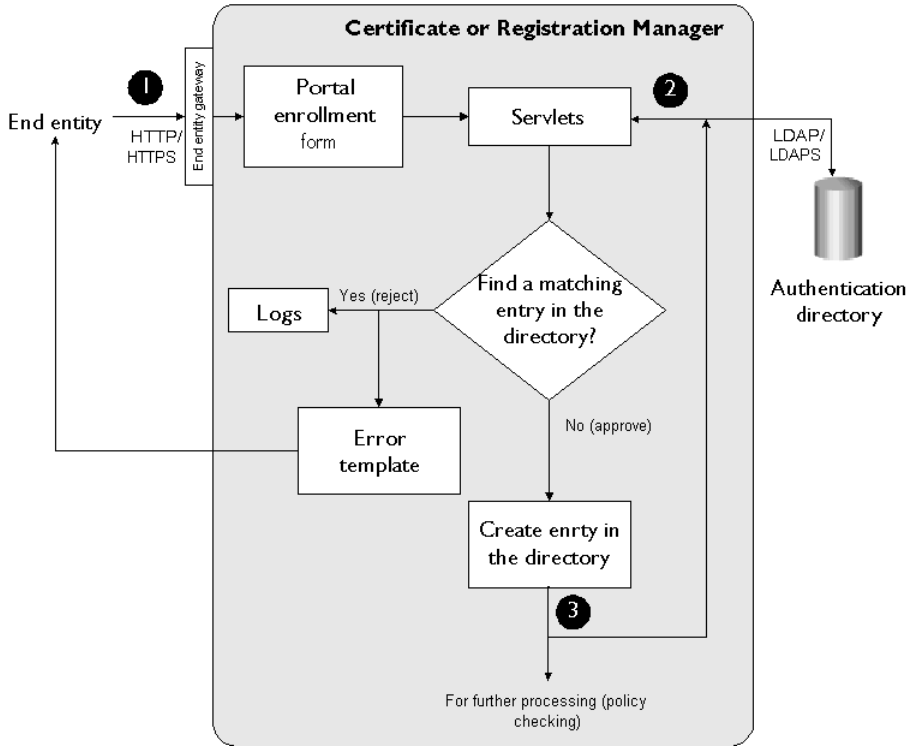
For example, if the HTTP form input contains data such as surname, common name, and phone number, the corresponding LDAP attributes would be set in the directory; for details, see "Enrollment Forms" on page 361. The server also uses a combination of these attributes (which you can specify using the `dnpattern` parameter defined in the module) to construct subject names for certificates.

Note that the portal authentication module by default uses the standard LDAP object class named `inetOrgPerson` to create and update user entries. The input fields defined in the default portal enrollment form correspond to the attributes defined in this object class as defined in Netscape Directory Server 4.x. The module is capable of reading and writing these attributes only. However, you can customize the module to accommodate all the fields supported by popular portals by extending the directory schema to include a new object class; you'll also be required to update the enrollment form to include attributes corresponding to the new object class. For guidelines on how to customize the module, check the sample located here:

```
<server_root>/cms_sdk/samples/authentication
```

Figure 10.8 illustrates how the portal authentication module works during certificate enrollment.

Figure 10.8 Portal authentication of an end user



These are the steps shown in Figure 10.8:

1. In the portal enrollment form, the end user enters registration information, such as a user name or ID, password, first name, last name, and mailing address, and submits the request to the server.

2. When the server receives the request, it verifies that the required fields contain appropriate information, for example, the values entered in the Password and Confirm Password fields match. Next, the server looks up the directory that is configured for authenticating portal service users for a matching user name.

    - If the server finds a matching user name in the directory, it rejects the request, logs an error message, and sends a rejection notification to the end user.

- If the server fails to find a matching user name in the directory, it uses the registration information to create a user entry for the new user and add relevant attributes. The server also retrieves information required to construct the subject name for the certificate.

If, for some reason, the directory to which the server binds for authenticating the user ID and password is unavailable, the server returns an LDAP error code and writes it to the log. When troubleshooting any authentication failures, if you see a log entry with an LDAP error code, look up the corresponding error code at this URL:

```
http://help.netscape.com/kb/server/970303-9.html
```

A sample log entry with an LDAP error code is shown below:

```
28/Jun/1999:18:40:25 -0700] conn=0 op=7 RESULT err=32 tag=101 nentries=0 etime=0]
```

3. Next, the server subjects the certificate request to policy processing; for details, see "Policies" on page 485.

   - If the request fails any of the configured policies, the server rejects the request, logs an error message, and sends a rejection notification to the end user. Note that if this happens, the user won't be able to reregister using the same user name.

   - If the request passes all the configured policies, the server issues the end user a certificate.

     The end user gets the certificate, which, if the server is configured to do so, is delivered to the email address specified in the request or in the directory; for information on configuring a Certificate Manager or Registration Manager to send automated notifications, see "Notifications of Certificate Issuance to End Entities" on page 450.

PortalEnroll Module

The Java class that implements the portal authentication plug-in module is identified as follows:

`com.netscape.certsrv.authentication.PortalEnroll`

- In the configuration file, the module is identified as follows: `auths.impl.PortalEnroll.class=com.netscape.certsrv.authentication.PortalEnroll`
- In the CMS window, the module is identified as follows: `PortalEnroll`

Figure 10.9 shows how configurable parameters for the `PortalEnroll` module are displayed in the CMS window.

Figure 10.9 Parameters defined in the PortalEnroll module

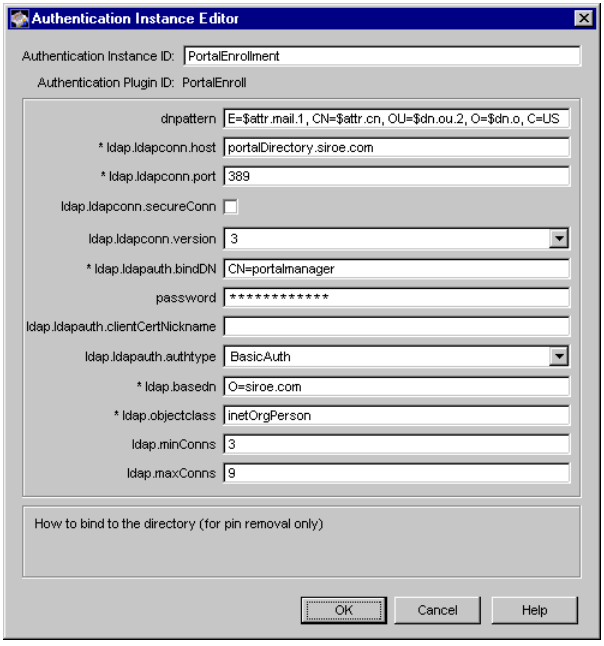


Table 10.5 gives details about each of these parameters and their values.

Table 10.5 Description of parameters defined in the PortalEnroll module

Parameter	Description
`dnpattern`	Specifies a string representing a subject name pattern to formulate from the directory attributes and entry DN. Permissible values: Any valid DN string composed from standard DN attributes, which must be separated by commas; see "DNs in Certificate Management System" on page 1157. The syntax is illustrated in the following example: `E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US` This sample configuration specifies that the subject name should be formulated as follows: • `E` = the first `mail` LDAP attribute value in the user's entry • `CN` = the (first) `cn` LDAP attribute value in the user's entry • `OU` = the second `ou` value in the user's entry DN • `O` = the (first) `o` value in the user's entry DN • `C` = the string `US` If this parameter value is empty or not set, the server uses `E=$attr.mail, CN=$attr.cn, O=$dn.o, C=$dn.c` as the DN pattern. This default DN pattern works well with Netscape Communicator and other browsers. For Communicator, if you leave out `E=` in end-user certificates, S/MIME may not work correctly (assuming lack of other extensions in the certificate). Also, if `C=` and `O=` are left out, certificate display looks strange in Communicator (when the Display Certificate button is clicked).
`ldap.ldapconn.host`	Specifies the host name of the portal directory. Permissible values: The name must be in the `<machine_name>.<your_domain>.<domain>` form. Example: `portalDirectory.siroe.com`

Table 10.5 Description of parameters defined in the PortalEnroll module (Continued)

Parameter	Description
`ldap.ldapconn.port`	Specifies the TCP/IP port at which the portal directory listens to requests from Certificate Management System. Permissible values: Any valid port number. Example: `389`
`ldap.ldapconn.secureConn`	Specifies the type—SSL or non-SSL—of the port at which the portal directory listens to requests from Certificate Management System. • Check the box if the port is an SSL (HTTPS) port. If your portal directory is configured for SSL-enabled communication (with or without SSL client authentication), choose this option. • Leave the box unchecked if the port is a non-SSL (HTTP) port. If your portal directory is configured for basic authentication, choose this option (default).
`ldap.ldapconn.version`	Specifies the LDAP protocol version. Permissible values: `2` or `3`. • `2` specifies LDAP version 2. If your portal directory is based on Netscape Directory Server 1.x, choose `2`. • 3 specifies LDAP version 3. For Directory Server versions 3.x and later, choose `3` (default). Example: `3`
`ldap.ldapauth.bindDN`	Specifies the user account to bind as in order to create and update user entries in the portal directory. It is recommended that you create and use a separate user account that has permission to create user entries and modify user attributes in the directory. For example, don't use the directory manager's entry as it has privileges to modify the entire directory content. Permissible values: A valid bind DN. Example: `CN=Portal Registration Manager`
`password`	Specifies the password associated with the DN specified by the `ldap.ldapauthbindDN` parameter. Permissible values: As applicable.

Table 10.5 Description of parameters defined in the PortalEnroll module (Continued)

Parameter	Description
`ldap.ldapauth.clientCertNickname`	Specifies the nickname or the friendly name of the certificate to be used for SSL client authentication to the portal directory. Make sure that the certificate is valid and has been signed by a CA that is trusted in the portal directory's certificate database, and that the portal directory's `certmap.conf` file has been configured to correctly map the certificate to a DN in the directory. Permissible values: Enter the name of a currently valid CMS certificate, for example, its SSL server certificate. Example: `Server-Cert`
`ldap.ldapauth.authtype`	Specifies the authentication type—basic authentication or SSL client authentication—required to communicate with the portal directory. Permissible values: `BasicAuth` or `SslClientAuth`. • `BasicAuth` specifies basic authentication. If you choose this option, be sure to enter the correct values for `ldap.ldapauth.bindDN` and `password` parameters; the server uses the DN from the `ldap.ldapauth.bindDN` attribute to bind to the directory (default). • `SslClientAuth` specifies SSL client authentication. If you choose this option, be sure to set the value of the `ldap.ldapconn.secureConn` parameter to `true` and the value of the `ldap.ldapauth.clientCertNickname` parameter to the nickname of the certificate to be used for SSL client authentication. Example: `BasicAuth`
`ldap.basedn`	Specifies the base DN for searching the portal directory—the server uses the value of the `uid` field from the HTTP input (what a user enters in the enrollment form) and the base DN to construct an LDAP search filter. Permissible values: Any valid DN string of up to 255 characters. Example: `O=siroe.com`
`ldap.objectclass`	Specifies the object class to modify or update in the portal directory. Permissible values: Must be `inetOrgPerson` for the default portal enrollment form; see "Enrollment Forms" on page 361. Example: `inetOrgPerson`

Table 10.5 Description of parameters defined in the PortalEnroll module (Continued)

Parameter	Description
`ldap.minConns`	Specifies the minimum number of connections permitted to the directory. Permissible values: `1` to `3` Example: `2`
`ldap.maxConns`	Specifies the maximum number of connections permitted to the directory. Permissible values: `3` to `10` Example: `10`

Certificate-Based Enrollment

Certificate Management System supports certificate-based enrollment for browser certificates. End users can use preissued certificates to authenticate to the server in order to enroll for certificates. Below are two deployment scenarios that explain the usefulness of certificate-based enrollment.

- You have deployed a client that can generate dual key pairs and you want to issue dual certificates (one for *signing* and another for *encrypting* data) to your users. You also want to make sure that users put their key materials only on hardware tokens.

  One way to achieve this would be to initialize hardware tokens in bulk and preload them with dual certificates issued by Certificate Management System for dual key pairs. You generate these certificates with some generic-looking common names, for example, `hardwaretoken1234`. This way, there's no one-to-one relation between users and the hardware tokens initially. Once the tokens are ready, you make them available to users by some means, for example, from a vending-machine-like box in the break room. Basically, a user can get and use any pre-initialized and certificate-loaded hardware token.

  Next, each user uses the randomly-picked token to enroll (strictly speaking, renew) for a pair of certificates that have a subject name derived from their LDAP attribute values; the certificates will be issued for the existing key pairs preloaded into the token, but now the key pairs will be associated with the user's identity.

- You want users use the signing certificate already in their possession to get an encryption certificate.

  For example, assume you have deployed Certificate Management System and have issued single certificates (for single key pairs) to users. Recently, you deployed a client application (such as Netscape Personal Security Manager) that is capable of generating dual key pairs. Your CMS installation includes the Data Recovery Manager, but you weren't using it until now because you didn't have clients that were capable of generating dual-key pairs. Now, you want your users to use their signing certificates as authentication tokens to request another certificate that they'll use for encrypting data.

To enable you to configure Certificate Management System for certificate-based enrollment, the following three enrollment forms are provided:

- `CertBasedDualEnroll.html`

  This form enables end users to request dual certificates—one for signing another for encryption—by submitting pre-issued certificates as authentication tokens; when a user enrolls for a certificate, the server verifies the CA that has issued the certificate the user uses for authentication, uses the configured directory to formulate subject names for the new certificates, and issues the certificates.

- `CertBasedEncryptionEnroll.html`

  This form is provided as a sample. It enables end users to request *encryption* certificates by submitting pre-issued certificates as authentication tokens; when a user enrolls for a certificate, the server verifies the CA that has issued the certificate the user uses for authentication, uses the configured directory to formulate the subject name for the new certificate, and issues the certificate.

- `CertBasedSingleEnroll.html`

  This form is provided as a sample. It enables end users to request *signing* certificates by submitting pre-issued certificates as authentication tokens; when a user enrolls for a certificate, the server verifies the CA that has issued the certificate the user uses for authentication, uses the configured directory to formulate the subject name for the new certificate, and issues the certificate.

Note that all three enrollment forms by default work with the directory-based authentication module, named `UidPwdDirAuth`, explained in "Directory-Based Authentication" on page 325. You can use the certificate-based enrollment

forms with any of the authentication modules, for example, directory- and PIN-based or NIS-server based authentication modules. However, this would require you to add the necessary hidden fields or variables to enrollment form that's provided for the corresponding authentication module; check Table 10.6 on page 363 to figure out which enrollment form works with which module.

In general, the following three hidden variables distinguish certificate-based enrollment forms from other enrollment forms:

- `certauthEnroll`—this variable specifies whether certificate-based enrollment is turned `on` or `off`.
- `certauthEnrollType`—this variable specifies one of the three certificate-based-enrollment types: `dual`, `single`, or `encryption`; `dual` specifies that the enrollment request is for dual certificates; `single` specifies that the enrollment request is for a signing certificate; and `encryption` specifies that the enrollment request is for an encryption certificate.

  Note that choosing `dual` would require a client that's capable of generating dual key pairs.
- `doSslAuth`—this variable specifies whether the server should request the client for SSL client authentication. You must set the value of this parameter to `on` and make sure that the port number specified in the authentication instance is an SSL port.

Before modifying a form, be sure to take a look at the default certificate-based enrollment forms. Also check the customization-related information for enrollment forms in "Customizing End-Entity and Agent Interfaces" on page 905.

In addition to the enrollment forms, a policy plug-in named `IssuerConstraints` is also provided; see "Issuer Constraints Policy" on page 509. This plug-in allows you to configure the server to recognize the CA that issues the certificates that your users will use for authentication purposes; you need this policy to ensure that the CA issues certificates only to those users who present a valid certificate during enrollment. Note that in the current implementation, the CA that issues the new certificates must be the same as the one that issues the certificates users will use for authentication. That is, the issuer DN in the authentication certificate must match the issuer DN specified in the policy configuration.

Here are a few things to keep in mind:

- Enrollment requests for dual certificates must be submitted directly to the Certificate Manager; the Registration Manager doesn't support generation of dual certificates.
- The Certificate Manager provides a bulk-enrollment interface, which can be used to preload keys and certificates on hardware tokens before distributing them to users for certificate enrollment. For details, see "Bulk Enrollment Interface" on page 978.
- When using certificate-based enrollment, the `IssuerConstraints` policy must be enabled and configured to check the CA (its issuer DN) in certificates users will use to authenticate to the server. Also, the value assigned to the `issuerDN` parameter must match the issuer DN of the CA that was used to generate hardware tokens in bulk.
- Enabling certificate-based enrollment creates one link, named `Certificate`, under the list of user-enrollment links in the end-entity enrollment interface. By default, the link points to the `CertBasedDualEnroll.html` form. If you want to use either of the other two forms, `CertBasedEncryptionEnroll.html` or `CertBasedSingleEnroll.html`, you should associate the `Certificate` link to the form you want to use or add more links to the `index.html` file.

General guidelines to set up certificate-based enrollment (for dual certificates) are as follows:

On the server side you need do the following:

- Customize the enrollment form you want your users to use for enrollment.
- Enable the appropriate enrollment option, such as directory-based enrollment or NIS-server based enrollment. Be sure to configure the authentication module to compose the desired DN pattern.
- Enable the Key Usage extension policy explained in "Key Usage Extension Policy" on page 618.

  Take a look at the key-usage policy rule named `ClientCertKeyUsageExt` and see if it needs any modifications. For example, to get a signing-only certificate, you need to turn off `keyEncipherment` and `dataEncipherment` bits of the extension; similarly, to get an encryption-only certificate, you may need to turn off the `digitalSignature` bit of the extension.

- Configure the `IssuerRule` policy with the correct issuer DN and set the predicate expression so that the rule is applied to client certificates only.

On the client side, you need to do the following:

- Install drivers for the hardware tokens you want to use during bulk generation of key pairs and corresponding certificates with generic subject names.
- If you want to issue dual certificates, install a client that can generate dual key pairs; for example, Netscape Communicator (version 4.7 or later) with Netscape Personal Security Manager.

Enrollment Forms

The end-entity interface of the Certificate Manager and the Registration Manager include default HTML forms for all the authentication methods—manual and automated—supported by the server.

Enrollment forms can be categorized into two types, depending on the authentication method they support.

- Manual enrollment forms—these forms work with the built-in manual authentication module, enabling users to request all types of certificates such as client certificates, server certificates, object-signing certificates, CA certificates, and so on. Manual enrollment for end users requires them to enter information such as name, email ID, department, organization, and the state and the country in which the organization is located, and submit the request for a personal certificate. Manual enrollment for server certificates requires the server administrator to paste the certificate signing request (in the PKCS#10 format) from the server into the specified area in the enrollment form; see “Certificate Issuance to Servers” on page 1081.

  Because the Certificate Manager or Registration Manager cannot verify the information an end user or administrator enters against anything, it builds the certificate request based on the user input and puts the request in the agent queue for approval.
- Automated enrollment forms—these forms work with the corresponding plug-in module, enabling users to request certificates by authenticating to the configured repository, for example an LDAP directory or NIS directory.

All enrollment forms are accessible from the Enrollment tab of the End Entity Services interface. Note that by default the Enrollment tab lists only those forms that are associated with the manual enrollment method (it does not list the forms provided for the automated-enrollment methods). However, when you create an instance of any of the authentication modules provided for automated end-user enrollment—for example, directory-based or NIS-server based authentication—a link to the corresponding form is automatically created under the Browser section of the Enrollment tab. Instructions for enabling automated end-user enrollments is covered in "Setting Up Authentication for End-User Enrollment" on page 387.

Before asking users to use any of the enrollment forms, you should review the form and make the appropriate changes to the form content. For example, some of the forms include instructions for administrators and you should delete those lines. Files for all enrollment forms are located here:

```
<server_root>/cert-<instance_id>/web/ee
```

For basic instructions to customize any of the enrollment forms, see "Step 5. Set Up the Enrollment Interface" on page 400. For advanced information on customizing the HTML forms, including how to make changes to the embedded Javascript functions, see "Agent and End-Entity Interfaces" on page 893.

Figure 10.10 shows the End Entity Services interface of a Certificate Manager with the manual enrollment form for end users selected.

Figure 10.10 End Entity Services interface of a Certificate Manager

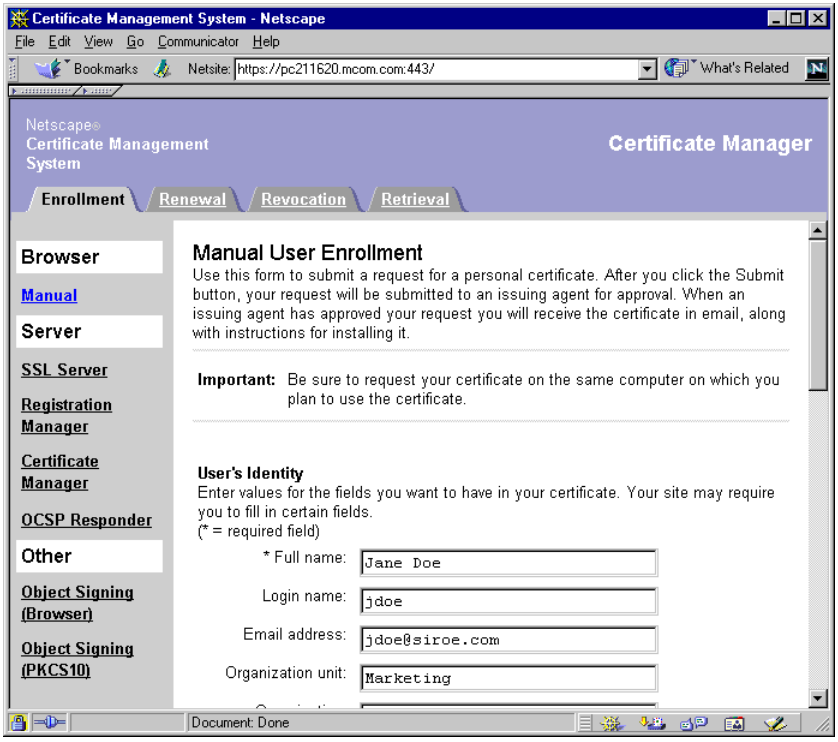


Table 10.6 lists the forms that correspond to the menu options in the Enrollment tab of the End Entity Services interface of the Certificate Manager and Registration Manager.

Table 10.6 Default forms for end-entity enrollment

Menu link and form filename	Description
Browser (this section lists menu options for end-user enrollments)	
Manual (`ManUserEnroll.html`)	End users can use this form to request SSL client and S/MIME certificates. Requests submitted using this form get queued for agent approval.
Directory (`DirUserEnroll.html`)	This form works with the `UidPwdDirAuth` module, enabling end users to request SSL client and S/MIME certificates by entering their user IDs and passwords for the directory; the server verifies this information against the configured directory and issues the certificate.

Table 10.6 Default forms for end-entity enrollment (Continued)

Menu link and form filename	Description
Directory and PIN (`DirPinUserEnroll.html`)	This form works with the `UidPwdPinDirAuth` module, enabling end users to request SSL client and S/MIME certificates by entering their user IDs, passwords, and PINs for the configured directory; the server verifies this information against the specified directory and issues the certificate.
NIS (`NISUserEnroll.html`)	This form works with the `NISAuth` module, enabling end users to request SSL client and S/MIME certificates by entering their NIS user IDs and passwords for the configured NIS server.
Portal (`PortalEnrollment.html`)	This form works with the `PortalEnroll` module, enabling end users to register for an online service and at the same time submit a request for a personal certificate. Note that the form models the standard LDAP object class `inetOrgPerson`, which has many useful attributes that can be used in a real portal deployment. As a part of registration, a user is required (by the portal authentication module) to supply a user ID and password for user ID validation and a first and last name for user registration. Entering information in other fields are optional; the server retrieves the rest of the information needed to construct the subject name for the certificate from the directory. As explained in "Portal Enrollment" on page 348, if the user ID is unique, the server issues a certificate and registers the user automatically. To protect the privacy of a user's password, the server turns it in to a SHA-1 or MD5 hashed password before storing it in the directory.
Certificate (`CertBasedDualEnroll.html`)	This form by default works with the `UidPwdDirAuth` module, enabling end users to request dual certificates (one for signing another for encryption) by submitting pre-issued certificates as authentication tokens; the server verifies the CA that has issued the certificate, uses the configured directory to formulate the subject names for the new certificates, and issues the certificate. Note that the link appears only if you create an instance of the `UidPwdDirAuth` module and if the port number specified in the instance configuration is an SSL port. For details, see "Certificate-Based Enrollment" on page 357.

Table 10.6 Default forms for end-entity enrollment (Continued)

Menu link and form filename	Description
Server (this section lists menu options for SSL server, Registration Manager, Certificate Manager, and OCSP Responder enrollments)	
SSL Server (`ManServerEnroll.html`)	Server administrators can use this form to request SSL server certificates for SSL-enabled servers, such as Netscape Administration Server and Directory Server. Requests submitted using this form get queued for agent approval.
Registration Manager (`ManRAEnroll.html`)	Registration Manager administrators can use this form to request a *signing certificate* for a Registration Manager; see "Signing Key Pair and Certificate" on page 230. Requests submitted using this form get queued for agent approval.
Certificate Manager (`ManCAEnroll.html`)	Certificate Manager administrators can use this form to request *CA signing certificates* for Certificate Managers functioning as subordinate CAs; see "CA Signing Key Pair and Certificate" on page 226. Requests submitted using this form get queued for agent approval. Only the Certificate Manager provides this form.
OCSP Responder (`OCSPResponder.html`)	Server administrators can use this form to enroll for an OCSP responder certificate. Requests submitted using this form get queued for agent approval.
Other (this section lists menu options for object signing enrollments)	
ObjectSigning (PKCS10) (`ObjSignPKCS10Enroll.html`)	Server administrators can use this form to enroll for a certificate (by submitting the request in the PKCS #10 format) that allows them to sign objects, such as Java applets. Requests submitted using this form get queued for agent approval.
ObjectSigning (Browser) (`ManObjSignEnroll.html`)	End users and administrators can use this form to enroll for a certificate that allows them to sign objects, such as Java applets. Requests submitted using this form get queued for agent approval. Note that when issuing an object signing certificate to Microsoft IE, if you need to generate a `.CER` and a `.PVK` file for use by Microsoft signcode tool, follow the instructions in "Generating Files Required By Third-Party Object Signing Tools" on page 366.

Generating Files Required By Third-Party Object Signing Tools

When issuing an object-signing certificate to Microsoft IE, Certificate Management System can generate a certificate (`.CER`) and a private key (`.PVK`) files for use by Microsoft `signcode` tool or any third-party sign tools that rely on these files. For the server to generate these files, you must edit the default form provided for requesting an object-signing certificate for browsers.

To generate the private-key file:

1. Go to this directory: `<server_root>/cert-<instance_id>/web/ee`
2. Locate the file named `ManObjSignEnroll.html`.
3. Open it in an editor.
4. Search for this line:

   ```
   Enroll.GenKeyFlags = 1        ' key exportable
   ```

5. Type the following line below it:

   ```
   Enroll.PVKFilename = "<pvk_file_path>"
   ```

   Your changes should look like this:

   ```
   ...
   Enroll.GenKeyFlags = 1        ' key exportable
   Enroll.PVKFilename = "<pvk_file_path>"
   szCertReq = Enroll.createPKCS10(szName, "1.3.6.1.5.5.7.3.2")
   ...
   ```

6. Replace `<pvk_file_path>` with the absolute path, including the filename, to the directory in which you want the private key file created; for example, `"C:\myKey.PVK"`. Be sure to use the `.PVK` extension and to enclose the path in double quotes.
7. Optionally, you may further edit the form to include a text field for entering the file path.
8. Save your changes.
9. Now use the form to issue an object-signing certificate.

If your users need to generate Software Publishing File (SPC) files for their object-signing certificates, you should ask them to use the Microsoft tool named `cert2spc`. The SPC file enables them to execute commands such as this:

```
signcode -spc myCert.spc -v myKey.pvk file.exe
```

Here's how a user can create a SPC file for an object-signing certificate:

1. Open a web browser window.
2. Go to the End Entity Services interface.
3. Locate the object-signing certificate for which you want to create the SPC file.
4. Scroll down the page (that shows the certificate information in detail) to find the certificate in base-64 encoded format. It looks like this:

   ```
   -----BEGIN CERTIFICATE-----
   MIICJzCCAZCgAwIBAgIBAzANBgkqhkiG9w0BAQQFADBCMSAwHgYDVQQKExdOZXRzY2FwZSBDb21td
   W5pY2F0aW9uczngjhnMVQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk4MDgyNzE5MDAwMFoXDTk5M
   DIyMzE5MDAwMnbjdgngYoxIDAeBgNVBAoTF05ldHNjYXBlIENvbW11bmljYXRpb25zMQ8wDQYDVQQ
   LEwZQZW9wbGUxFzAVBgoJkiaJkIsZAEBEwdzdXByaXlhMRcwFQYDVQQDEw5TdXByaXlhIFNoZXR0e
   TEjMCEGCSqGSIb3DbndgJARYUc3Vwcml5YUBuZXRzY2FwZS5jb20wXDANBgkqhkiG9w0BAQEFAANL
   ADBIAkEAoYiYgthgtbbnjfngjnjgnagwJjAOBgNVHQ8BAf8EBAMCBLAwFAYJYIZIAYb4QgEBAQHBA
   QDAgCAMA0GC2pWZWUm7waHEtdbo9vSpbJkXTM/2GhWbsO5vLdeOxrPGxihkHgV/yyggiufr4s3az
   -----END CERTIFICATE-----
   ```

5. Create an ASCII file named `cert.b64`.
6. Copy and paste the base-64 encoded certificate blob, including the marker lines `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` to the file.
7. Convert the text-based certificate to its DER-encoded format using the ASCII to Binary tool, explained in

   For example, the command
   `<server_root>/bin/cert/tools/AtoB cert.b64 cert.der`

   converts the base-64 encoded certificate in the `cert.b64` file to its DER-encoded format and writes the DER-encoded certificate to a file named `cert.der`.

8. Next, use the Microsoft tool named `cert2spc` to convert the DER-encoded certificate to SPC format. For example, the command `cert2spc cert.der cert.spc` converts the DER-encoded certificate in the `cert.der` file to its SPC format and writes the certificate to a file named `cert.spc`.

For additional information, check these links:

- `http://www.lantimes.com/ltparts/connect/shoptalk1.htm`
- `http://www.thawte.com/certs/developer/msauthenticode.html`
- `http://www.drh-consultancy.demon.co.uk/pkcs12faq.html`

Chapter

11

Using the PIN Generator Tool

For Netscape Certificate Management System (CMS) to use the plug-in module for *directory-based authentication with PINs*, your authentication directory must contain unique PINs for each end entity to whom you intend to issue a certificate. To aid you in generating PINs for end-entity entries in a directory, Certificate Management System provides a command-line tool called the *PIN Generator*. This tool allows you to generate unique PINs for entries in an LDAP-compliant user directory. The tool stores these PINs (as hashed values) in the same directory against the corresponding user entries, and it copies the PINs to a text file, from which you can deliver the PINs to end entities by an appropriate, secure means.

This chapter explains how to use the PIN Generator. The chapter has the following sections:

Locating the PIN Generator Tool

You can find the PIN Generator at this location:

```
<server_root>/bin/cert/tools/setpin.exe
```

The setpin Command

You run the PIN Generator by entering the `setpin` command and its arguments in a command shell and monitoring the output in the shell window. This section gives the syntax for the `setpin` command and its arguments. For instructions on generating PINs and storing them in your authentication directory, see "Step 2. Set Up the Directory for PIN-Based Enrollment" on page 389.

Command-Line Syntax

Use the following command in a Unix or DOS command shell:

```
setpin [arguments]
```

```
setpin [optfile=filename] [param1] [param2]
```

Arguments

The `setpin` command takes the following arguments and options:

```
setpin
   [host=<host_name> [port=<port_number>]]
   [certdb=<path> nickname=<certificate_nickname> |
   "binddn=<user_id>" bindpw=<bind_password> [SSL=yes | no]]
   [objectclass=<objectclass_to_add>]
   [attribute=<attribute_name_for_pins>]
   ["filter=<LDAP_search_filter>"]
   [input=<file_name>]
   [length=<PIN_length> | minlength=<minimum_PIN_length>
   maxlength=<maximum_PIN_length>]
```

```
[gen=RNG-alpha | RNG-alphanum | RNG=printableascii]
[case=upperonly]
[hash=sha1 | md5 | none]
[output=<file_name>]
[clobber]
[write]
[saltattribute=<LDAP_attribute_to_use_for_salt_creation>]
[debug]
```

A description for each argument follows:

- `[host=<host_name> [port=<port_number>]]`

  `<host_name>` specifies the LDAP directory to connect to.

  `<port_number>` specifies the TCP/IP port to bind to; the default port number is the default LDAP port, 389.

- `[certdb=<path> nickname=<certificate_nickname> | "binddn=<user_id>" bindpw=<bind_password> [SSL=yes | no]]`

  Use this argument to specify how the tool should connect to the directory: whether to use basic authentication, SSL, or SSL with client authentication. By default, SSL is not used (`SSL=no`).

  — If you want the tool to use basic authentication, you must turn off SSL (`SSL=no`) and enter the bind DN and password information. Do not enter values for the remaining options.

  — If you want the tool to use SSL, you must turn on SSL (`SSL=yes`) and enter the bind DN and password information. Do not enter values for the remaining options.

  — If you want the tool to use SSL with client authentication, you must turn on SSL (`SSL=yes`) and enter the nickname of the certificate to use for SSL client authentication and the path to the certificate database that contains this certificate. You don't have to provide the bind DN and password.

  `<path>` specifies the path to the certificate database containing the client authentication certificate to use for SSL client authentication. If you provide the path to the certificate database (`cert7.db` in Netscape Directory

Server), it is assumed that the LDAP directory has been set up for SSL-based access. You must also specify the certificate for SSL client authentication to the directory.

`<certificate_nickname>` specifies the nickname of the certificate to use for SSL client authentication to the directory. The tool looks for this certificate in the database specified by the `path` parameter value. If you want the tool to use a client certificate residing on a token or smart card to access the directory, prefix the nickname with the word *module* (`module:nickname`); then a PKCS #11 module will be used.

`<user_id>` specifies the user ID that has read and write permission to the LDAP directory; the PIN Generator binds to the directory as this user.

`<bind_password>` specifies the password for the user ID that has read and write access to the LDAP directory.

If the bind password is not given at the command line, the tool prompts for it.

- `[objectclass=<objectclass_to_add>]`

  Use this argument to specify the object class, if any, the tool should add to the authentication directory. By default it is `pinPerson`.

- `[attribute=<attribute_name_for_pins>]`

  Use this argument to specify the authentication directory attribute to which PINs should be published. If you don't specify an attribute, it defaults to `pin`, the new attribute added to the authentication directory schema; see "Step 2. Set Up the Directory for PIN-Based Enrollment" on page 389.

- `["filter=<LDAP_search_filter>"]`

  Use this argument to filter those DNs in the directory for which the tool should generate PINs. For information on how to specify filters, see the information available at this URL:

  `http://developer.netscape.com/docs/manuals/dirsdk/capi/search.htm`

- `[input=<file_name>]`

  Use this argument to specify the name of the file that contains the list of DNs to process. Using this argument is optional. If you do, the tool compares the filtered DNs to the ones specified by the input file and generates PINs for only those DNs that are also in the file.

- `[length=<PIN_length> | minlength=<minimum_PIN_length> maxlength=<maximum_PIN_length>]`

  Use this argument to specify the exact number or a range of characters that a PIN must contain. The PINs can be either a fixed length or generated to be between two values (x,y) inclusive (x,y>0).

  `<PIN_length>` specifies the exact length for the PINs. For example, if you want PIN length to be eight characters, enter `8`. PIN length must be an integer greater than zero.

  `<minimum_PIN_length>` specifies the minimum length for the PINs. For example, if you want PIN length to be at least six characters, enter `6`.

  `<maximum_PIN_length>` specifies the maximum length for the PINs. For example, if you want PIN length to be nine characters at the most, enter `9`.

- `[gen=RNG-alpha | RNG-alphanum | RNG-printableascii]`

  Use this argument to specify the type of characters for PINs. The characters in the password can be constructed out of alphabetic characters (`RNG-alpha`), alphanumeric characters (`RNG-alphanum`), or any printable ASCII characters (`printableascii`).

- `[case=upperonly]`

  Use this argument with the `gen` parameter. If you do, the case for all alphabetic characters is fixed to uppercase only; otherwise, the case is mixed. Restricting alphabetic characters to uppercase reduces the overall combinations for the password space significantly.

- `[hash=sha1 | md5 | none]`

  Use this argument to specify the message digest algorithm the tool should use to hash the PINs before storing them in the authentication directory. If you want to store PINs as SHA-1 or MD5 hashed values in the directory, be sure to specify an output file for storing PINs in plain text. You will need the PINs in plain text for delivering them to end entities.

  `sha1` produces a 160-bit message digest. This option is used by default.

  `md5` produces a 128-bit message digest.

  `none` does not hash the PINs.

- `[output=<file_name>]`

  Use this argument to specify the absolute path to the file to which the tool should write the PINs as it generates them; this is the file to which the tool will capture the output.

If you don't specify a filename, the tool will write the output to the standard output. In any case, all the error messages will be directed to the standard error.

- `[clobber]`

  Use this argument to specify whether the tool should overwrite preexisting PINs, if any, associated with a DN (user). If specified, the tool overwrites the existing PINs with the one it generates. Otherwise, it leaves the existing PINs as they are.

- `[write]`

  Use this argument to specify whether the tool should write PINs to the directory. If specified, the tool writes PINs (as it generates) to the directory. Otherwise, the tool does not make any changes to the directory.

  For example, if you want to check PINs—that the PINs are being given to the correct users and that they are conforming to the length and character-set restrictions—before updating the directory, do not specify this option. You can check the PINs before updating the directory by looking at the output file; for details, see "Output File" on page 379.

- `[saltattribute=<LDAP_attribute_to_use_for_salt_creation>]`

  Use this argument to specify the LDAP attribute the tool should use for salt creation. If you specify an attribute, the tool integrates the corresponding value of the attribute with each PIN, and hashes the resulting string with the hash routine specified in the hash argument.

  If you don't specify this argument, the DN of the user is used. For details, see "How PINs Are Stored in the Directory" on page 380.

- `[debug]`

  Use this argument to specify whether the tool should write debugging information (to the standard error). If `debug=attrs` is specified, the tool writes much more information about each entry in the directory.

- `[testpingen=<count>]`

  Use this argument to test the pin-generation mode.

  `<count>` specifies the total number (in decimal) of PINs to be generated for testing purposes.

- `[optfile]`

  Use this argument to specify that the tool should read in options (one per line) from specified file; this option enables you to put all the arguments in a file, instead of typing them on the command line.

Example

The following command generates PINs for all entries that have the `CN` attribute (in their distinguished name) defined in an LDAP directory named `laiking` that is listening at port `19000`. The PIN Generator binds to the directory as user `DirectoryManager` and starts searching the directory from the node `dn=o=airius.com` in the directory tree. The tool overwrites the existing PINs, if any, with the new ones.

```
setpin host=lailing port=19000 "binddn=CN=directoryManager"
bindpw=password "filter=(cn=*)" basedn=o=airius.com clobber
write
```

How the Tool Works

The Pin Generator allows you to generate PINs for user entries in an LDAP-compliant directory and update the directory with these PINs. To run the `setpin` command, you need at a minimum to specify the following:

- The host name (`host`) and port number (`port`) of the LDAP server
- The bind DN (`binddn`) and password (`bindpw`)
- An LDAP filter (`filter`) for filtering out the user entries that require PINs

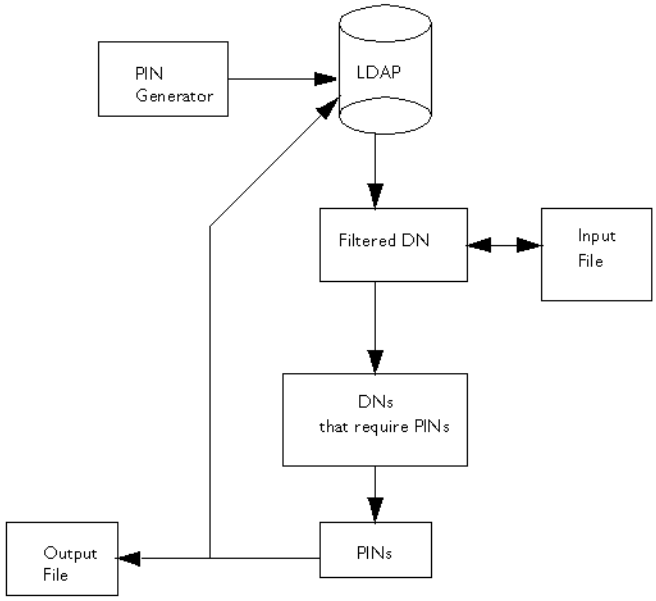
For example:

```
setpin host=laiking port=19000 "binddn=CN=Directory Manager"
bindpw=netscape "filter=(ou=employees)" basedn=o=siroe.com
```

This command, if run, will query the directory for all the entries that match the filter criteria, which in this case is all users belonging to an organizational unit (`ou`) called `employees`. For each entry matching the filter, information is printed out to standard error. Additionally, to the standard output or the file named in output; see “Output File” on page 379.

You can also provide the tool with an input argument using the `input` option. The argument must be in the form of an ASCII file of pre-prepared DNs and PINs (see Figure 11.1). Note that the input file isn't a substitute for the LDAP directory entries; the filter attribute must still be provided. If an input file is provided, the tool updates only those filtered attributes that match the ones in the input file. For more information about the input file, see "Input File" on page 378.

Figure 11.1 Using an input and output file for the PIN-generation process



Examples of output follow:

```
Processing: cn=QA Managers,ou=employees,o=airius.com
 Adding new pin/password
dn:cn=QA Managers,ou=employees,o=airius.com
pin:lDWynV
status:notwritten
```

```
Processing: cn=PD Managers,ou=employees,o=airius.com

 Adding new pin/password

dn:cn=PD Managers,ou=employees,o=airius.com

pin:G69uV7

status:notwritten
```

Because the PIN Generator makes a lot of changes to your directory, it is important that you specify the correct filter; otherwise, you may change the wrong entries. As a safeguard, a `write` option is provided that you use to enable writing to the directory after you verify the output; the tool doesn't make any changes to the directory until you specify the `write` option on the command line.

The output also contains the status of each entry in the directory. It can be one of the values specified in Table 11.1.

Table 11.1 PIN Generator status

Exit code	Description
`notwritten`	Specifies that the PINs were not written to the directory, because the `write` option was not specified on the command line.
`writefailed`	Specifies that the tool made an attempt to modify the directory, but the write operation was unsuccessful.
`added`	Specifies that the tool added the new PIN to directory successfully.
`replaced`	Specifies that the tool replaced an old PIN with a new one (the `clobber` option was specified).
`notreplaced`	Specifies that the tool did not replace the old PIN with a new one (the `clobber` option was not specified).

If a PIN already exists for a user, it will by default not be changed if you run the `setpin` command a second time. This is so that you can generate PINs for new users without overwriting PINs for users who have previously been notified of their PINs. If you want to overwrite a PIN, you should use the `clobber` option.

Once you are sure that the filter is matching the right users, you should run the `setpin` command again with the `write` option, and with `output` set to the name of the file to capture the unhashed PINs. This output file is in the same format as the input file. For details about the output file, see "Output File""Output File" on page 379.

Input File

The PIN Generator can receive a list of DNs to modify in a text file specified by the `input=<file_name>` argument. If you specify an input file, the tool compares the DNs it filtered from the LDAP directory with the ones in the input file, and updates only those DNs that matched the ones in the input file.

The purpose of the input file is multifold. It enables you to provide the Pin Generator with an exact list of DNs to modify. Via the input file, you can also provide the PIN Generator with PINs (in *plain text* format) for all DNs or for specific DNs.

The following examples explain why you might want to use the input file:

- Assume that you have set PINs for all entries in the user directory. Two new users joined your organization and you updated the directory with new users' information. For the new users to get certificates, the directory must contain PINs. And you want to set PINs for just those user entries without making changes to any of the other user entries. Instead of constructing a complex LDAP filter to filter out just these two entries, you can construct a general filter, put the two users' DNs in the input file, and run the PIN Generator.

- Assume that you want your users to use their social security numbers as PINs. You can enter users' social security numbers as PINs in the input file, and the PIN Generator will store them as hashed values in the directory.

The format of the input file is the same as that of the output file (see "Output File" on page 379), with the omission of the status line. In the input file, you can choose to specify PINs for all the DNs in the file, for specific DNs, or for none of the DNs. If the PIN attribute is missing for a DN, the tool automatically generates a random PIN.

For example, you can set up your input file to look like this:

```
dn:cn=user1, o=siroe
<blank line>
dn:cn=user2, o=siroe
<blank line>
...
dn:cn=user3, o=siroe
```

You can also provide PINs, in plain-text format, for the DNs in the input file, which is then hashed according to the command-line arguments. For example, you can set up your input file to look like this:

```
dn:cn=user1, o=siroe
pin:pl229Ab
<blank line>
dn:cn=user2, o=siroe
pin:9j65dSf
<blank line>
...
dn:cn=user3, o=siroe
pin:3knAg60
<blank line>
```

Note You cannot provide hashed PINs to the tool.

Output File

The PIN Generator can capture the output to a text file specified by the `output=<file_name>` argument.

The captured output will contain a sequence of records and will be in the following format:

```
dn: <user_dn>1
pin: <generated_pin>1
status: <status>1
<blank line>
```

```
dn: <user_dn>2
pin: <generated_pin>2
status: <status>2
<blank line>
...
dn: <user_dn>n
pin: <generated_pin>n
status: <status>n
<blank line>
```

where

`<user_dn>` is a distinguished name that matched the specified DN pattern (specified by the DN filter) or that was in the input file (the DN file). By default, the delimiter is `";"` or the character defined on the command line.

`<generated_pin>` is a string of characters with either fixed or variable length, dependent on parameters passed into the command.

`<status>` is one of the values specified in Table 11.1 on page 377.

The first line in each record will always be the distinguished name. The subsequent lines, for `pin` and `status`, are optional. The record ends with a blank line. The end of line (EOL) sequence is as follows:

Windows NT `\r\n`

Unix `\n`

How PINs Are Stored in the Directory

Each PIN is concatenated with the corresponding user's LDAP attribute named in the `saltattribute` argument. If this argument is not specified, the DN of the user is used. Then, this string is hashed with the hash routine specified in the hash argument (the default selection is SHA-1).

Then, one byte is prepended to indicate the hash type used. Here's how the PIN gets stored:

```
byte[0] = X
```

The value of `X` depends on the hash algorithm chosen during the PIN generation process:

`X=0` if the hash algorithm chosen is `SHA-1`.

`X=1` if the hash algorithm chosen is `MD5`.

`X=45` if the hash algorithm chosen is `none`.

```
byte[1...] = hash("DN"+"pin")
```

The PIN is stored in the directory as a binary value, not as a base-64 encoded value.

Exit Codes

The PIN Generator returns exit codes to the shell window; for a list of codes, see Table 11.2. If you plan on automating the PIN-generation process, exit codes are useful in programming shell scripts.

Table 11.2 Exit codes returned by the PIN Generator

Exit code	Description
`0`	Indicates that PIN generation was successful; that is, PINs are set for all the DNs in the specified directory.
`2`	Indicates that the tool could not open the certificate database specified by the `certdb` parameter.
`3`	Indicates that the tool could not locate the certificate specified by the `nickname` parameter in the specified certificate database.
`4`	Indicates that the tool could not bind to the directory as the user specified by the `binddn` parameter (over SSL).
`5`	Indicates that the tool could not open the output file specified by the `output` parameter.
`7`	Indicates an error parsing command-line arguments.
`8`	Indicates that the tool could not open the input file specified by the `input` parameter.

Table 11.2 Exit codes returned by the PIN Generator (Continued)

Exit code	Description
9	Indicates that the tool encountered an internal error.
10	Indicates that the tool found a duplicate entry in the input file specified by the `input` parameter.
11	Indicates that the tool didn't find the salt attribute, specified by the `saltattribute` parameter, in the directory.

Chapter

12

Configuring Authentication for End Users

Netscape Certificate Management System (CMS) provides a customizable authentication component that supports various methods for authenticating end users. This chapter explains how to configure a Certificate Manager and Registration Manager to use specific authentication plug-in modules for authenticating end users during certificate enrollment.

Before reading this chapter, you should have read the chapters "Introduction to Authentication" on page 305 and "Authentication Modules for End-Entity Enrollment" on page 319. In particular, you should be familiar with the various plug-in modules for authenticating end users that come with Certificate Management System. If you are not, see "Overview of Authentication Modules" on page 320.

The chapter has the following sections:

Authentication Management

You can manage end-user authentication in two ways:

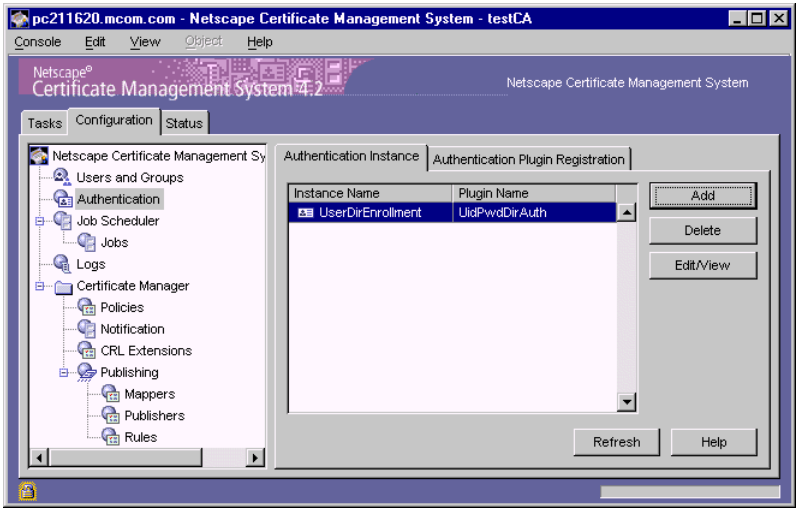
- By making changes to the authentication-specific parameters from the CMS window, explained in “Authentication Management From the CMS Window” on page 384
- By editing the parameters in the configuration file, explained in “Authentication Parameters in the Configuration File” on page 386

The recommended method is to use the CMS window.

Authentication Management From the CMS Window

Figure 12.1 shows the CMS window, which provides the user interface for configuring and managing the end-user authentication.

Figure 12.1 End-user authentication information in the CMS window



In the navigation tree of the CMS window, you will find a single `Authentication` object. Selecting this object shows the authentication modules and instances currently recognized by this instance of Certificate Management System. From this window you can do the following:

- Configuration of currently registered authentication plug-in modules
- Registration of new (custom) authentication plug-in modules

The sections that follow describe the parts of the window from which you carry out these operations.

Authentication Instance Tab

The Authentication Instance tab lists the currently configured authentication instances, so that you can manage them at a single place. From this tab you can perform the following operations:

Add. The add operation shows a list of registered authentication plug-in modules from which you can select the one you want to configure. When you save the changes, Certificate Management System creates the new authentication instance and displays it in the list of authentication instances. For instructions, see "Step 4: Add an Authentication Instance" on page 395.

Delete. The delete operation allows you to remove unwanted authentication instances from the CMS configuration. For instructions, see "Deleting an Authentication Instance" on page 410.

Edit/View. The edit operation allows you to view and modify the configuration parameter values of currently configured authentication instances. For instructions, see "Modifying an Authentication Instance" on page 411.

Authentication Plugin Registration Tab

The Authentication Plugin Registration tab lists the currently registered authentication plug-in modules for the selected CMS instance and gives you access to the window from which you can register new authentication plug-in modules. On this tab you will find the names of registered modules listed on the left and the path to the Java class that implements the module listed on the right.

You can perform the following operations from this tab:

Register. This operation allows you to register a new authentication plug-in module. When you save the changes, Certificate Management System loads the authentication plug-in implementation and displays it in the list of registered modules. For instructions, see "Registering an Authentication Module" on page 414.

Delete. This operation allows you to remove unwanted authentication plug-in modules from the CMS framework. For instructions, see "Deleting an Authentication Module" on page 416.

Authentication Parameters in the Configuration File

The sample configuration file shown on page 89 illustrates how authentication-specific information appears in the configuration file. Keep the following points in mind:

- All authentication-specific information, such as names of registered authentication plug-in modules and any configured instances, appears in the Authentication section of the configuration file.
- Each registered authentication plug-in module is identified by its implementation name and the corresponding Java class.
- Each configured instance of an authentication module is identified by the name or ID you specified when creating it.
- You can create multiple instances out of an implementation; each instance must have a unique name.
- The name of an authentication instance must be used in the corresponding enrollment form so that the server is able to determine the authentication method during end-user enrollment. For details, see "Setting Up Authentication for End-User Enrollment" on page 387.

To change the configuration by editing the configuration file, follow the instructions in "Changing the Configuration by Editing the Configuration File" on page 86.

Managing Authentication Instances

This section explains how to use the CMS window to do the following:

- Setting Up Authentication for End-User Enrollment
- Deleting an Authentication Instance
- Modifying an Authentication Instance

For information on adding or changing authentication-specific information in the configuration file, see "Authentication Parameters in the Configuration File" on page 386.

Setting Up Authentication for End-User Enrollment

To set up a Certificate Manager and Registration Manager to authenticate end users based on a specific criteria, follow these steps:

- Step 1: Find the Required Information
- Step 2. Set Up the Directory for PIN-Based Enrollment (required for PIN-based enrollment only)
- Step 3. Enable the PIN Present Policy (required for PIN-based enrollment only)
- Step 4: Add an Authentication Instance
- Step 5. Set Up the Enrollment Interface
- Step 6. Enable End-Entity Interaction
- Step 7. Turn on Automated Notification
- Step 8. Test Your Authentication Setup
- Step 9. Deliver PINs to End Users (required for PIN-based enrollment only)

Note If you do not configure a Certificate Manager or Registration Manager to use any of the registered authentication plug-in modules, the server uses manual authentication for end-user enrollment. This means that all end-user enrollment requests are queued for agent approval. For more information, see "Manual Authentication" on page 323.

Step 1: Find the Required Information

Before setting up a Certificate Manager or Registration Manager to use a specific authentication method:

- Determine the authentication module you want to use. To find out about the modules that are installed with the server, see "Overview of Authentication Modules" on page 320. If you want to develop and use a custom plug-in module, be sure to read "Developing Custom Authentication Modules" on page 417.

  — If you decided to use the directory-based authentication module, note the authentication directory credentials, such as the host name, port number, base DN, the user entry to bind as and the corresponding password, the DN pattern to retrieve from the directory to construct certificate subject names, LDAP version number, and minimum and maximum number of connections permitted. Check Table 10.2 on page 329 to get the complete list of parameters for which you'll be asked to provide values for.

  — If you decided to use the directory- and PIN-based authentication module, note the authentication directory credentials, such as the host name, port number, based DN, the user entry to bind as and the corresponding password, LDAP version number, and minimum and maximum number of connections permitted. Check Table 10.3 on page 334 to get the complete list of parameters for which you'll be asked to provide values for.

    Next, read "Using the PIN Generator Tool" on page 369, determine the options you want to use to generate PINs, and construct the command for generating the PINs. Note that the `optfile` option enables you to put all the arguments in a file (instead of typing the arguments at the command prompt) and then point the tool to read arguments from the file.

— If you decided to use the NIS server-based authentication module, note the NIS server host name and domain name. If you have an LDAP directory deployed and want to use that for formulating the certificate subject names, note the directory-specific information. Check Table 10.4 on page 344 to get the complete list of parameters for which you'll be asked to provide values for.

— If you decided to use the portal authentication module, note the LDAP directory-specific information. Check Table 10.5 on page 354 to get the complete list of parameters for which you'll be asked to provide values for.

- Determine the enrollment form you want your users to use. For guidance, check "Enrollment Forms" on page 361.

The next step depends on the authentication module you chose:

- If you decided to use the directory- and PIN-based authentication module, go to the next step, "Step 2. Set Up the Directory for PIN-Based Enrollment" on page 389.
- For all other modules, skip to "Step 4: Add an Authentication Instance" on page 395.

Step 2. Set Up the Directory for PIN-Based Enrollment

Go through this step only if you want to configure the server to use the directory- and PIN-based authentication method (with or without PIN removal).

To set up a directory for PIN-based authentication method:

- Step A. Check the Directory for User Entries
- Step B. Update the Directory
- Step C. Prepare the Input File
- Step D. Run the Command Without the Write Option
- Step E. Check the Output File
- Step F. Run the Command Again with the Write Option

Step A. Check the Directory for User Entries

Before you run the PIN Generator tool, make sure that the directory you intend to use for generating PINs has been populated with end-user entries that require PINs. It is also a good idea to confirm with that directory's administrator that all pending directory requests have been processed before the PIN Generator starts its operation.

Step B. Update the Directory

By default, the PIN Generator modifies the `pin` attribute in a directory's user entry (see "Arguments" on page 370). Because this attribute is not part of the standard `organizationalPerson`, you need to add it—that is, create a new object class (named `pinPerson`) in your authentication directory's schema.

In general, you need to update the `slapd.user_at.conf` file to include the `pin` attribute and the `slapd.user_oc.conf` file to include the object-class definition. The modified schema should look similar to this:

```
attribute pin bin

objectclass pinPerson
   superior organizationalPerson
   allows
      pin
```

In addition, if you want to make use of the PIN-removal feature—that is, remove a user's PIN from the directory after Certificate Management System successfully authenticates that user and thus prevent the user from enrolling for another certificate—ACIs must be set up on the directory to prevent end users from creating new PINs for themselves. To do this, you need to create an entry for a *PIN manager* user with read-write permission to the `pin` attribute.

The PIN Generator tool comes with a configuration file, named `setpin.conf`, which enables you to automate the process of updating the authentication directory with changes required for setting up PIN-based authentication. The configuration file is located in this directory:

```
<server_root>/bin/cert/tools
```

To make the required schema changes and add an entry for the PIN manager user:

1. Go to this directory: `<server_root>/bin/cert/tools`
2. Open the `setpin.conf` file in a text editor.
3. Follow the instructions outlined in the file and make the appropriate changes.

   Typically, you will need to update the Directory Server's host name, Directory Manager's bind password, and PIN manager's password.
4. Run the `setpin` command with its `optfile` option pointing to the `setpin.conf` file (`setpin optfile=setpin.conf`).

   The tool modifies the schema with a new attribute (by default, `pin`) and a new object class (by default, `pinPerson`), creates a `pinmanager` user, and sets the ACI to allow only the `pinmanager` user to modify the `pin` attribute.

   If the tool is able to update the directory with all the changes, you should see a status-update message, similar to the sample shown below:

```
Adding attribute: ( pin-oid NAME 'pin' DESC 'User Defined Attribute'
SYNTAX '1.3.6.1.4.1.1466.115.121.1.5' SINGLE-VALUE )
 .. successful

Adding objectclass: ( pinPerson-oid NAME 'pinPerson' DESC 'User
Defined ObjectClass' SUP 'top' MUST ( objectclass ) MAY ( aci $ pin )
 .. successful

Adding user: cn=pinmanager,o=siroe.com
 .. successful

modifying ACI for: ou=people,o=siroe.com
 .. successful
```

Step C. Prepare the Input File

This step is optional.

If you want to generate PINs for specific user entries or want to provide your own PINs, use an input file (to provide the tool with such information). For information on constructing an input file, see "Input File" on page 378.

Step D. Run the Command Without the Write Option

Run the `setpin` command without the `write` option. Be sure to include the `output` option and bind to the directory as the PIN manager user. For details, see "The setpin Command" on page 370.

The tool will write PINs to the specified output file; no changes are made to the directory yet. This will give you the opportunity to check the PINs (by looking at the output file) before updating the directory.

To run the command:

1. Open a terminal window.
2. Go to this directory: `<server_root>/bin/cert/tools`
3. Run the `setpin` command with the appropriate arguments; be sure to point the `optfile` option to the file you've created (and not to the `setpin.conf` file).

Step E. Check the Output File

Check the output file to be sure it contains PINs for your users; the output should look similar to the one specified in "Output File" on page 379.

Next, verify that the tool has assigned PINs to the correct users and that the PINs conform to the length and character-set restrictions you specified. If the output isn't what you want, run the command again with appropriate arguments. Repeat the process until the output file shows the results you want.

Step F. Run the Command Again with the Write Option

When you are sure about the results, run the command again (with exactly the same arguments) with the `write` option and the `output` option. The tool stores the hashed PINs in the directory. For information on how PINs are stored in the directory, see "How PINs Are Stored in the Directory" on page 380.

Use the output file for delivering PINs to users after you complete setting up the required authentication method; see "Step 9. Deliver PINs to End Users" on page 410.

Step 3. Enable the PIN Present Policy

This step is required for PIN-based enrollment with PIN removal only.

To deploy PIN-based enrollment with PIN removal effectively, the PIN present policy must be enabled and configured properly. This policy, if enabled, checks the authentication directory for a user's PIN when an end user requests a certificate and issues the certificate only if the PIN exits in the directory—thus preventing users from getting a certificate in the absence of a PIN. If you are not familiar with the role this policy plays in PIN-based enrollment, see "PIN Present Policy" on page 514.

Unlike some of the other policy rules, Certificate Management System does not create an instance of the PIN present policy rule during installation. If you created this rule after installation, you can configure the server to use that rule. The instructions below explain how to create a new rule:

1. Log in to the CMS window for the Certificate Manager (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Certificate Manager, and then select Policies.

   The Policy Rules Management tab appears. It lists currently configured policy rules.
4. Click Add

   The Policy Plugin Implementation window appears.

5. Select `PinPresentConstraints` and click Next.

   The Policy Rule Editor window appears. It lists the configuration information required for this policy rule.

6. Enter the appropriate information.

   Policy Rule ID. Type a unique name that will help you identify the rule. Be sure to formulate the name using any combination of letters (`aA` to `zZ`), digits (`0` to `9`), an underscore (_), and a hyphen (`-`). For example, you can type `My_Policy_Rule` or `MyPolicyRule` as the instance name, but not `My Policy Rule`.

   enable. Check the box to enable the rule.

   predicate. Type the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank. Because the enrollment method is applicable for end user enrollments only, the typical predicate expression would be `HTTP_PARAMS.certType==client`. To form the correct predicate expression, see "Using Predicates in Policy Rules" on page 490.

7. Click OK.

   You are returned to the Policy Rules Management tab. If required, click the Reorder button and order the rules as appropriate. For details, see "Step 5. Reorder Policy Rules" on page 705.

Step 4: Add an Authentication Instance

Adding an authentication instance to the CMS configuration involves creating a new instance of an already registered plug-in module, assigning a unique name or ID to the instance, and entering appropriate values for the parameters that define the plug-in you want to create an instance of.

When naming an authentication instance (or rule), be sure to formulate the name using any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-); other characters and spaces are not allowed. For example, you can type `My_Auth_Rule` or `MyAuthRule` as the instance name, but not `My Auth Rule`.

Also note that when you add an authentication instance, the CMS configuration is updated with authentication-specific information only. The server does not associate the authentication instance you added with any of the end-user enrollment forms—that is, the end-user servlets that should use this authentication instance are not configured yet. You make this association by manually embedding the authentication instance name in the enrollment forms.

By default, the enrollment forms include authentication instance names listed in Table 12.1. Note that the authentication instances are not created by default; only the instance names are embedded in the forms for your convenience. If you create authentication instances with the default names, you can skip the step ("Step A. Associate the Authentication Instance With the Enrollment Form" on page 400) that explains how to update an enrollment form to associate it with the name of an authentication instance.

Table 12.1 Default authentication instance names embedded in end user enrollment forms

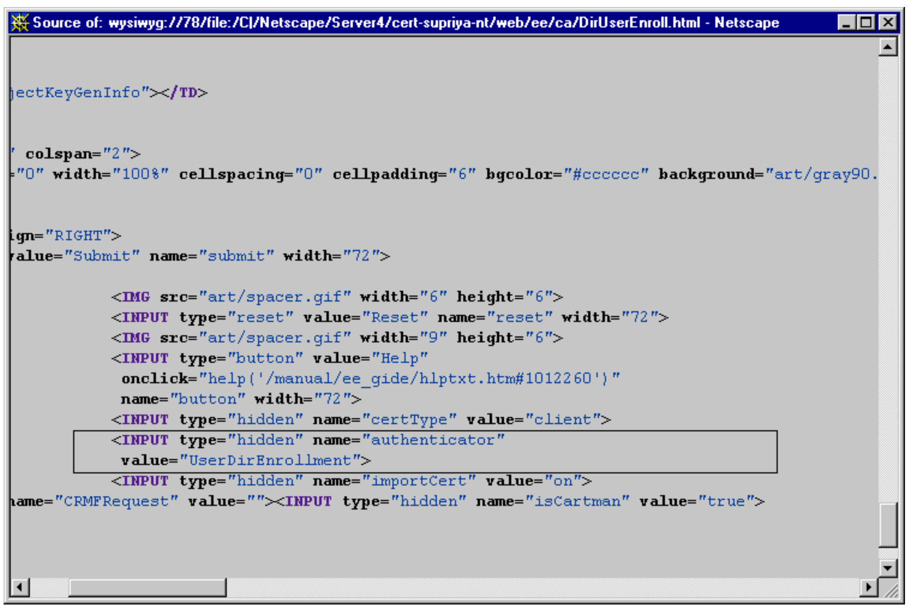
Enrollment form (filename)	Authentication instance name
Certificate-based enrollment for dual certificates (`CertBasedDualEnroll.html`)	`UserDirEnrollment`
Certificate-based enrollment for encryption certificates (`CertBasedEncryptionEnroll.html`)	`UserDirEnrollment`
Certificate-based enrollment for single certificates (`CertBasedSingleEnroll.html`)	`UserDirEnrollment`
Directory-based enrollment (`DirUserEnroll.html`)	`UserDirEnrollment`
Directory- and PIN-based enrollment (`DirPinUserEnroll.html`)	`PinDirEnrollment`

Table 12.1 Default authentication instance names embedded in end user enrollment forms (Continued)

Enrollment form (filename)	Authentication instance name
NIS server-based enrollment (`NISUserEnroll.html`)	`NISAuth`
Portal enrollment (`PortalEnrollment.html`)	`PortalEnrollment`

Figure 12.2 shows the default directory-based enrollment form configured to use an authentication instance named `UserDirEnrollment`.

Figure 12.2 Authentication information in the default directory-based enrollment form

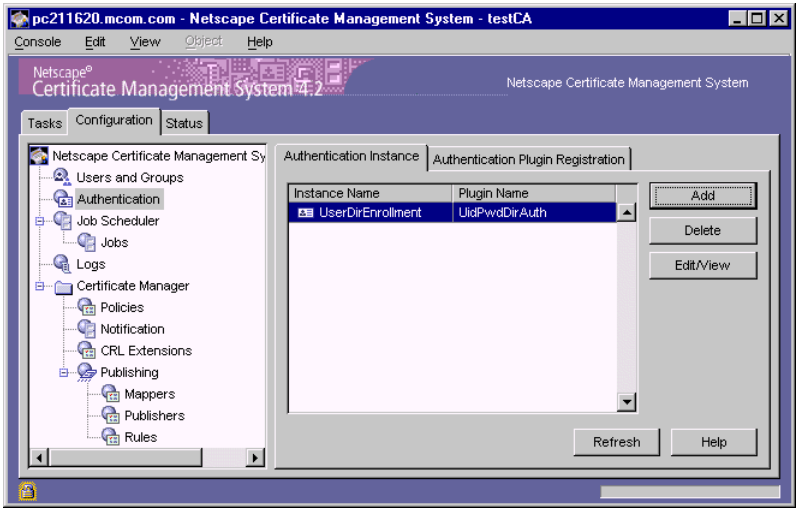


For information on locating and customizing the default end-entity forms, see "End-Entity Forms and Templates" on page 913.

To add an authentication instance to the CMS configuration:

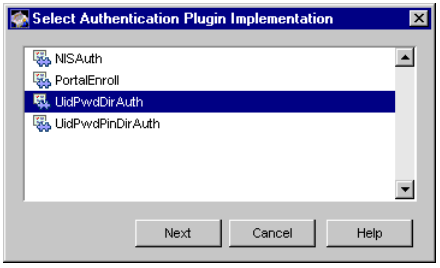
1. In the CMS window, select the Configuration tab.
2. In the navigation tree, select Authentication.

   The right pane shows the Authentication Instance tab, which lists any currently configured authentication instances.



3. Click Add.

   The Select Authentication Plugin Implementation window appears. It lists the currently registered authentication plug-in modules.



4. Select a plug-in module.

   The following choices are the ones provided by default with Certificate Management System (they are described in "Overview of Authentication Modules" on page 320). If you have registered any custom authentication plug-in modules, they too will be available for selection.

   UidPwdDirAuth. Select this if you want to use the directory-based authentication module; see "Directory-Based Authentication" on page 325.

   UidPwdPinDirAuth. Select this if you want to use the directory- and PIN-based authentication module (with or without PIN removal); see "Directory- and PIN-Based Authentication" on page 332.

   To configure Certificate Management System for PIN-based enrollment method, you must have completed "Step 2. Set Up the Directory for PIN-Based Enrollment" on page 389.

   NISAuth. Select this if you want to use the NIS server-based authentication module; see "NIS Server-Based Authentication" on page 340.

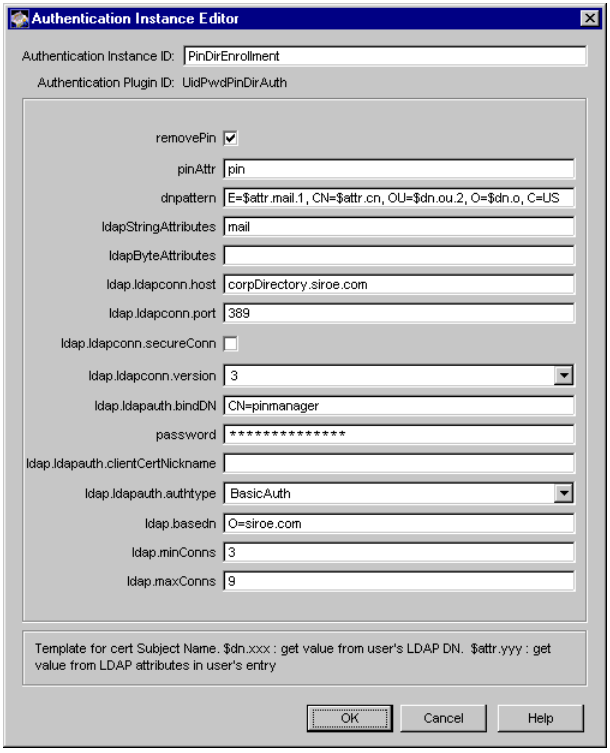
   PortalEnroll. Select this if you want to use the portal authentication module; see "Portal Enrollment" on page 348.

   For the purposes of this instruction, assume that you selected `UidPwdPinDirAuth`.

5. Click Next.

   The Authentication Instance Editor window appears. The Authentication Instance ID field shows the default instance name embedded in the associated enrollment form (see Table 12.1 on page 395). The

Authentication Plugin ID field shows the module you've chosen. The remaining fields list the configuration information required for this authentication instance.



6. If you don't want to use the default instance name, in the Authentication Instance ID field, type a unique name for this instance that will help you identify it.

   For the name, be sure to use an alphanumeric string with no spaces. If you chose to use a different name, be sure to edit the default name in the enrollment form in the next step, "Step 5. Set Up the Enrollment Interface" on page 400.

7. Fill in values for the remaining parameters.

8. Click OK.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. Don't restart the server yet; you can do this after you've made all the required changes.

Step 5. Set Up the Enrollment Interface

This step explains how to customize the end-entity interface for the enrollment method you've chosen for your users.

- Step A. Associate the Authentication Instance With the Enrollment Form
- Step B. Customize the Form
- Step C. Hook Up the Certificate-Based Enrollment Form (optional)
- Step D. Remove Unwanted Enrollment Options

Step A. Associate the Authentication Instance With the Enrollment Form

You can skip this step if, in the previous step, you chose the default instance name suggested in Table 12.1 on page 395. Otherwise, you must edit the enrollment form to associate the instance you added because Certificate Management System relies on the authentication instance name embedded in the enrollment form to determine the authentication method.

For the new authentication instance to work with end-entity enrollment forms, you must update the appropriate forms, as follows:

1. In the CMS host system, go to this directory:
   `<server_root>/cert-<instance_id>/web/ee`

2. Locate the file that corresponds to the authentication module you chose in "Step 4: Add an Authentication Instance" on page 395; use Table 12.1 for guidance.

3. Open the file in a text editor.

4. Locate the attribute that associates the authentication instance with the enrollment form.

   - In the default enrollment forms, locate this line:

```
<INPUT TYPE="HIDDEN" NAME="authenticator" VALUE="myAuthMgr">
```

- In the custom enrollment form, be sure to include the following line, and replace `myAuthMgr` with the name of the authentication instance you added.

```
<INPUT TYPE="HIDDEN" NAME="authenticator" VALUE="myAuthMgr">
```

5. Check the value assigned to `authenticator` attribute (the `VALUE=` field). Make sure that it is same as the name or ID you assigned to the authentication instance you created in Step 5. If it is different, replace it with the name of the authentication instance. For example, if you used `test_auth` as the instance name, the edited line should look like this: `<INPUT TYPE="HIDDEN" NAME="authenticator" VALUE="test_auth">`
6. Save your changes and close the file.

Step B. Customize the Form

You can make any other changes to meet your organization's requirements.

Step C. Hook Up the Certificate-Based Enrollment Form

This step is required if you want to use any of the certificate-based enrollment forms.

As explained in "Enrollment Forms" on page 361, Certificate Management System provides three forms for certificate-based enrollment:

- `CertBasedDualEnroll.html`
- `CertBasedEncryptionEnroll.html`
- `CertBasedSingleEnroll.html`

By default, the form named `CertBasedDualEnroll.html` is hooked up to the Enrollment tab of the end-entity interface. You can replace this form with either of the other two forms, `CertBasedEncryptionEnroll.html` and `CertBasedSingleEnroll.html`; you can do this by uncommenting the script relevant to either of the forms in the index file and by commenting out the script for `CertBasedDualEnroll.html`—thus, effectively unhook the old one and hook the new one.

To enable any of the certificate-based enrollment forms, follow these steps:

1. In the CMS host system, go to this directory:
   `<server_root>/cert-<instance_id>/web/ee`
2. Locate the `index.html` file.
3. Open the file in a text editor.
4. Follow instructions as appropriate:

   If you want to enable the `CertBasedDualEnroll.html` form, search for `CertBasedDualEnroll`. You should find a block of script like the following:

   ```
   count++;
   }
   if (http != 'true') {
   // this one is directory based cert-based
   if ( isAuthMgrEnabled("UidPwdDirAuth") ) {
   item = 'certBasedDualEnroll';
   menuItems[count] = top.EnrollMenu[count] =
   new menuItem(item, 'CertBasedDualEnroll.html',
   'Certificate');
   ```

   If you want to enable the `CertBasedEncryptionEnroll.html` form, search for `CertBasedEncryption`. You should find a block of script like the following:

   ```
   count++;
   }
  //   item = 'certBasedEncEnroll';
  //   menuItems[count] = top.EnrollMenu[count] =
  //      new menuItem(item,
  //        'CertBasedEncryptionEnroll.html', 'Certificate');
   ```

   Uncomment the lines and then add lines for using the automated enrollment module you configured the server with. Your edited lines should look similar to this:

   ```
   count++;
   }
   if (http != 'true') {
    // this one is directory based cert-based
   ```

```
if ( isAuthMgrEnabled("UidPwdDirAuth") ) {
  item = 'certBasedEncEnroll';
  menuItems[count] = top.EnrollMenu[count] =
     new menuItem(item, 'CertBasedEncryptionEnroll.html',
        'Certificate');
```

If you want to enable the `CertBasedSingleEnroll.html` form, search for `CertBasedSingle`. You should find a block of script similar to this:

```
   count++;
   }
//   item = 'certBasedSingleEnroll';
//   menuItems[count] = top.EnrollMenu[count] =
//      new menuItem(item, 'CertBasedSingleEnroll.html',
//        'Certificate');
```

Uncomment the lines and then add lines for using the automated enrollment module you configured the server with. Your edited lines should look like these:

```
count++;
}
if (http != 'true') {
  // this one is directory based cert-based
if ( isAuthMgrEnabled("UidPwdDirAuth") ) {
  item = 'certBasedSingleEnroll';
  menuItems[count] = top.EnrollMenu[count] =
     new menuItem(item, 'CertBasedSingleEnroll.html',
        'Certificate');
```

5. Make sure to comment out lines for any unused options.

6. If you're using any of the default modules, except for the one provided for the directory-based enrollment (`UidPwdDirAuth`), edit the following line to replace `UidPwdDirAuth` with the name of the module you're using.

   ```
   if ( isAuthMgrEnabled("UidPwdDirAuth") ) {
   ```

7. By default, a link named `Certificate` will be created under the Browser section. If you want to rename the link, replace `Certificate` in the following line with the new name:

   ```
   new menuItem(item, 'CertBasedDualEnroll.html',
   'Certificate');
   ```

8. Save your changes and close the file.

Step D. Remove Unwanted Enrollment Options

This step is optional.

By default, the Enrollment tab of the end-entity interface shows just one link, named `Manual`, under the Browser section. The `Manual` link opens a form that enables end users to request certificates manually. When you enable automated enrollment, that is, when you create an instance of any of the automated enrollment modules, a link for the corresponding form is automatically created under the Browser section. For example, if you create an instance of the directory-based authentication module, you will notice a new link named `Directory` under the Browser section.

If you don't want your end users to see the manual enrollment option, you can remove it from the Enrollment tab altogether. The steps below explain how to remove the Manual link from the Browser section of the Enrollment tab:

1. In the CMS host system, go to this directory:

   `<server_root>/cert-<instance_id>/web/ee`

2. Locate the `index.html` file.

3. Open the file in a text editor.

4. Locate the `initEnrollMenu` function (the `function initEnrollMenu()` line).

5. Remove or comment out lines that correspond to the Manual enrollment form.

   ```
   count++;
     item = 'manuser';
     menuItems[count] = top.EnrollMenu[count] =
        new menuItem(item, 'ManUserEnroll.html', 'Manual');
   ```

6. Save your changes and close the file.

7. Open a browser window.

8. Go to the end-entity interface and verify your changes.

Step 6. Enable End-Entity Interaction

You can configure end-entity interaction with a Certificate Manager or a Registration Manager, or with both. End entities cannot interact with a Data Recovery Manager directly; they must interact through a Certificate Manager or Registration Manager. By default, the Certificate Manager is configured for end-entity interaction; the Registration Manager is not configured for end-entity interaction.

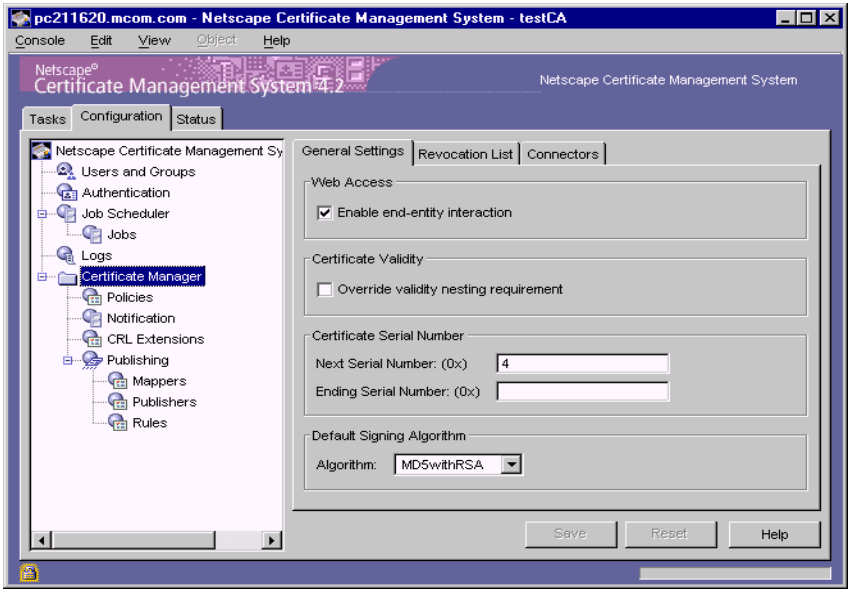
Depending on the subsystem you're configuring, follow the instructions in "Enabling End-Entity Interaction with a Certificate Manager" on page 405 or in "Enabling End-Entity Interaction with a Registration Manager" on page 407.

Enabling End-Entity Interaction with a Certificate Manager

To enable end-entity interaction with a Certificate Manager:

1. In the CMS window, select the Configuration tab.
2. In the navigation tree, select Certificate Manager.

   The General Setting tab appears.



3. In the Web Access section, check the "Enable end-entity interaction" option if you want end entities to be able to interact with the selected Certificate Manager via the HTTPS port; leave it unchecked to disable end-entity interaction with the server.

   Note that if you disable end-entity interaction, the Network tab still shows the HTTPS port and allows you to configure it (see "Configuring Port Numbers" on page 158). However, you should know that the server ignores this port.

4. In the Certificate Validity section, check the "Override validity nesting requirement" option, if you want the Certificate Manager to issue certificates with validity periods beyond that of its *CA signing certificate*; see "CA Signing Key Pair and Certificate" on page 226).

   If you leave the box unchecked and if the Certificate Manager (CA) finds a request with validity period extending beyond that of its CA signing certificate, it automatically truncates the validity period to end on the day the CA signing certificate expires. For example, if the CA signing certificate expires on June 10, 2004, any enrollment or renewal request with validity period beyond June 10, 2004 will have validity period truncated to end on June 10, 2004.

   Validity periods of certificates during enrollment is determined by the policy explained in "Validity Constraints Policy" on page 543. Similarly, validity periods of certificates during renewal is determined by the policy explained in "Renewal Validity Constraints Policy" on page 525.

5. In the Certificate Serial Number section, specify the serial number range for certificates issued by this Certificate Manager. The server assigns the serial number you enter in the "Next serial number" to the next certificate it issues and the number you enter in the "Last serial number" to the last certificate it issues.

   The serial number range enables you to deploy multiple CAs, balancing the number of certificates each CA issues. Note that the combination of an issuer name and a serial number uniquely identifies a certificate. To ensure that two distinct certificates issued by the same authority doesn't contain the same serial number, make sure the serial number range does not overlap among cloned CAs. (For information on cloning CAs, see *Netscape Certificate Management System Installation and Deployment Guide.*

6. In the Default Signing Algorithm section, select the signing algorithm the Certificate Manager should use for signing certificates. The choices are "MD2 with RSA," "MD5 with RSA," and "SHA1 with RSA," if the CA's signing key type is RSA and "SHA1 with DSA," if the CA's signing key type is DSA.

   Note that the signing algorithm specified in the Certificate Manager's policy configuration overrides the algorithm you select here. For information on a Certificate Manager's policy configuration, see "Signing Algorithm Constraints Policy" on page 533.

7. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

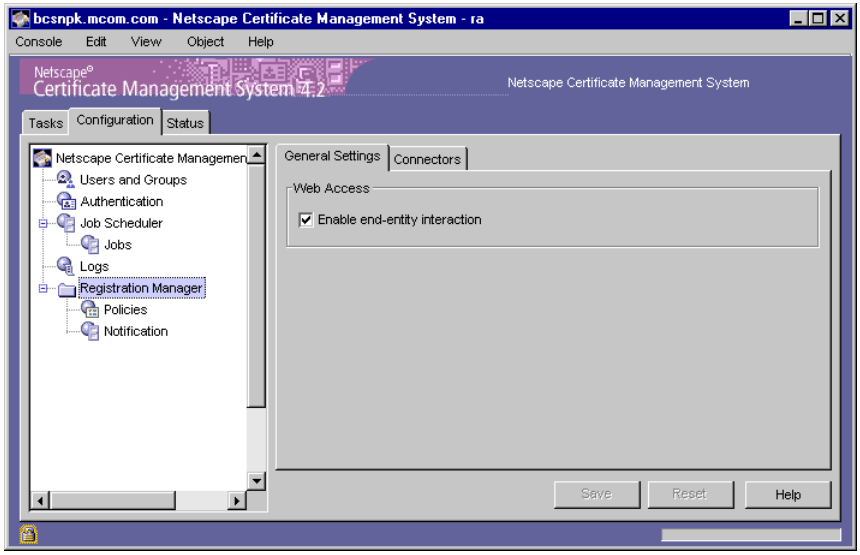
Enabling End-Entity Interaction with a Registration Manager

To enable end-entity interaction with a Registration Manager:

1. In the CMS window, select the Configuration tab.

2. In the navigation tree, select Registration Manager.

   The General Setting tab appears.



3. In the Web Access section, check the "Enable end-entity interaction" option if you want end entities to be able to interact with the selected Registration Manager via the HTTPS port; leave it unchecked to disable end-entity interaction with the server.

   Note that if you disable end-entity interaction, the Network tab still shows the HTTPS port and allows you to configure it (see "Configuring Port Numbers" on page 158). However, you should know that the server ignores this port.

4. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Step 7. Turn on Automated Notification

Both the Certificate Manager and the Registration Manager can send certificate-issuance notification to end users. For details on turning this feature on, see "Configuring a Subsystem to Send Notifications to End Entities" on page 451.

Step 8. Test Your Authentication Setup

To test whether your end users can successfully enroll for a certificate using the authentication method you've set up:

1. Open a web browser window.

2. Go to the end-entity interface for the enrollment authority you configured. The default URL is as follows:

   `https://<host_name>:<end_entity_HTTPS_port>` or
   `http://<host_name>:<end_entity_HTTP_port>`

3. In the Enrollment tab, open the enrollment form you customized.

4. Fill in all the values and submit the request.

5. The client prompts you to enter the password for your key database.

6. When you enter the correct password, the client generates the key pair.

   Do not interrupt the key-generation process. Upon completion of the key generation, the request is submitted to the server for certificate issuance. The server subjects the request to the currently configured policy rules and issues the certificate only if the request passes all the policy rules.

   Upon receipt of a notification about the certificate issuance, install the certificate in your browser.

7. Verify that the certificate is installed in the browser's certificate database; for example, in Communicator you can open the Security Info window and verify that the certificate is listed in there.

8. If you've set up the directory- and PIN-based authentication with PIN removal, reenroll for another certificate using the same PIN. Your request should get rejected.

9. If you've set up the portal enrollment, verify that an entry for the user is created in the directory. For example, you can point your browser to the portal directory and find out if an entry for the user for whom you requested the certificate exists.

   In the URL field, type `ldap://<host_name>:<port>/<base_dn>??sub?(uid=<user_id>)`, substituting `<host_name>` with the fully qualified host name of the Directory Server, `<port_number>` with the port number at which the Directory Server is listening to authentication requests from the Certificate Manager `<base_dn>` with the DN to start searching for the user's entry, and `<user_id>` with the ID of the user for whom you requested the certificate.

   For example, if the directory host name is `corpDirectory`, port number is `389`, base DN is `O=siroe.com`, and user's ID is `jdoe`, the URL would look like this:

   ```
   ldap://corpDirectory:389/O=siroe.com??sub?(uid=jdoe)
   ```

   In the resulting page, look for the user's credentials and verify that they match what you specified in the enrollment form. If you've configured Certificate Management System to publish certificates to the same directory ("Publishing Certificates and CRLs to a Directory" on page 786), you will be able to see the certificate-related information; it typically includes information such as the owner of the certificate, the CA that has issued the certificate, the serial number, the validity period, and the certificate fingerprint.

Step 9. Deliver PINs to End Users

This step is applicable for directory- and PIN-based authentication with or without PIN removal.

After you have confirmed that the PIN-based enrollment works (as it should), deliver the PINs to users so they can use them during enrollment. To protect the privacy of PINs, be sure to use a secure, out-of-band method for delivery. Here are a few suggested delivery methods:

- Encrypted email (S/MIME)—if your company has S/MIME mail set up, you can deliver PINs to users by encrypted mail.
- Mail—you can mail PINs to users, for example along with their pay stubs or slips.
- Personal delivery—you can arrange a secure means of delivering the password to the user, or ask the user to collect it from you in person.

Deleting an Authentication Instance

You can delete an authentication instance that you no longer need from the CMS configuration. If you delete an authentication instance, end users will fail to enroll for certificates using the associated enrollment form. If you want the form to work with another authentication instance, you must make the appropriate changes to the form; see "Step 5. Set Up the Enrollment Interface" on page 400.

To delete an authentication instance from the CMS configuration:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, click Authentication.

   The right pane shows the Authentication Instance tab, which lists currently configured authentication instances.
4. In the Instance Name list, select the instance you want to delete and click Delete.

5. When prompted, confirm the delete action.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Modifying an Authentication Instance

You can modify an authentication instance by editing its configuration parameter values; you cannot edit the name of an instance. To change the name of an instance, you need to create a new instance exactly like the instance you want to rename, except with a new name, and delete the old instance. For details, see "Step 4: Add an Authentication Instance" on page 395.

When you modify an authentication instance, the CMS configuration is updated to include the modifications. Because you are not changing the name of the authentication instance, you do not have to make any changes to the end-user servlet configuration.

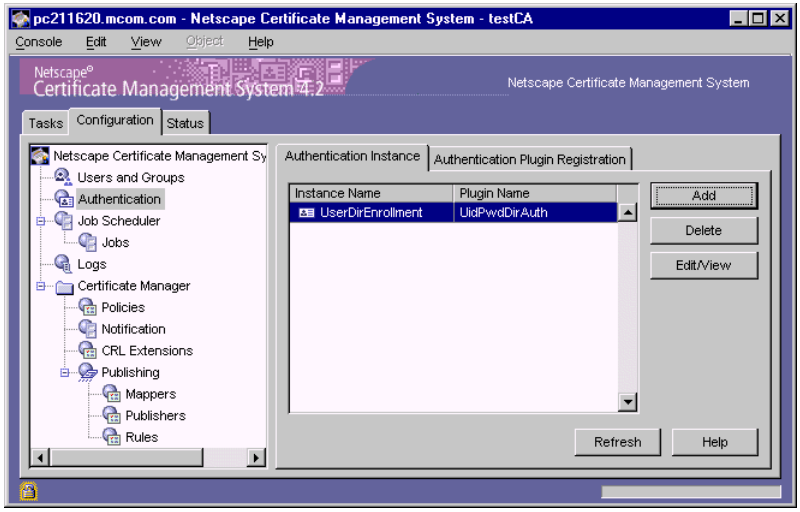
To modify an authentication instance in the CMS configuration:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

2. Select the Configuration tab.

3. In the navigation tree, click Authentication.

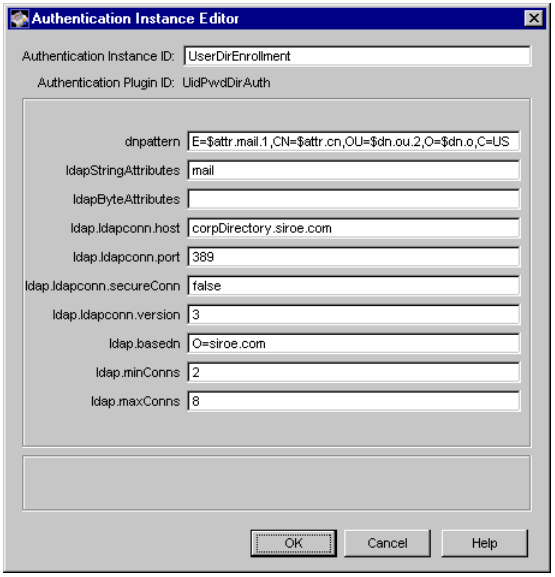
   The right pane shows the Authentication Instance tab, which lists configured authentication instances.



4. In the Instance Name list, select the instance you want to modify and click Edit.

   The Configure Authentication Instance Parameters window appears, showing the current configuration of this instance.

For the purposes of completing these instructions, assume you selected `UserDirEnrollment`.



5. Make changes as appropriate. If you need description for any of the parameters, click the Help button or check these:

   - For a description of configuration parameters of the `UidPwdDirAuth` module, see Table 10.2 on page 329.
   - For a description of configuration parameters of the `UidPwdPinDirAuth` module, see Table 10.3 on page 334.
   - For a description of configuration parameters of the `NISAuth` module, see Table 10.4 on page 344.
   - For a description of configuration parameters of the `PortalEnroll` module, see Table 10.5 on page 354.

6. Click OK.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Managing Authentication Plug-in Modules

This section explains how to use the CMS window to do the following:

- Registering an Authentication Module
- Deleting an Authentication Module

For information on adding or changing authentication-specific information in the configuration file, see "Authentication Parameters in the Configuration File" on page 386.

Registering an Authentication Module

You can register custom authentication plug-in modules from the CMS window. Registering a new authentication module involves specifying the name of the module and the full name of the Java class that implements the authentication interface. For example, you can add a Java class, named as follows, that implements an authentication module:
`com.netscape.certsrv.authentication.ssnAuth`

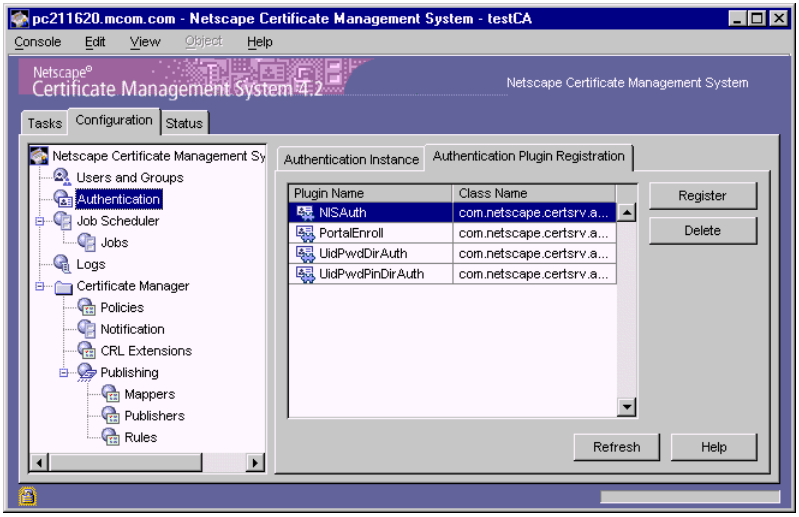
Before registering an authentication module, be sure to put the Java class for the module in the `classes` directory; see "Compiling and Installing Authentication Manager Plug-ins" on page 423.

To register an authentication module in the CMS authentication framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.

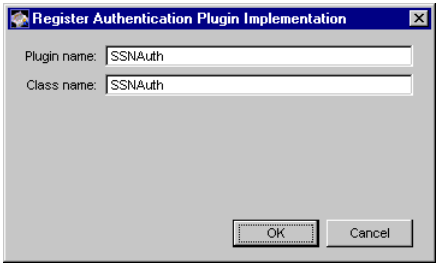
3. In the navigation tree, click Authentication, and in the right pane, click the Authentication Plugin Registration tab.

   The tab lists modules that are already registered.



4. Click Register.

   The Register Authentication Plugin Implementation window appears.



5. Specify which module you want to register:

   Plugin name. Type a name for the module.

   Class name. Type the full name of the class for this module—that is, the path to the implementing Java class. If this class is part of a package, be sure to include the package name. For example, if you are registering a class named `NISAuth` and if this class is in a package named `com.mycompany`, type `com.mycompany.NISAuth`.

6. Click OK.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Deleting an Authentication Module

You can delete an authentication plug-in module that you no longer need by using the CMS window. Before deleting a module, be sure to delete all the instances that are based on this module; see "Deleting an Authentication Instance" on page 410. You should also update the appropriate end-entity enrollment forms.

To delete an authentication module from the CMS authentication framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, click Authentication, and in the right pane, click the Authentication Plugin Registration tab.

   The tab lists the currently registered modules.
4. In the Plugin Name list, select the module you want to delete and click Delete.
5. When prompted, confirm the delete action.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Chapter

13

Developing Custom Authentication Modules

The authentication subsystem evaluates and authenticates the identity of an end entity that is requesting service from Netscape Certificate Management System (CMS). By default, the authentication subsystem supports various authentication mechanisms; these are explained in "Introduction to Authentication" on page 305.

If the authentication plug-in modules provided with Certificate Management System do not meet your requirements, you can write custom authentication plug-ins and plug them into the authentication framework of Certificate Management System.

This chapter provides an overview of the authentication subsystem architecture, discusses the interfaces for writing custom authentication plug-ins (called *authentication managers*), and explains how to implement custom authentication.

The chapter has the following sections:

Note To customize authentication, you must be familiar with programming in Java and the Netscape Certificate Management System 4.x software development kit (SDK).

Authentication Subsystem Architecture

The authentication subsystem of Certificate Management System permits a flexible number of authentication managers for end-entity authentication. This architecture is illustrated in Figure 13.1. An authentication manager (`authMgr`) is a configured instance of an authentication plug-in implementation.

Figure 13.1 Authentication subsystem architecture

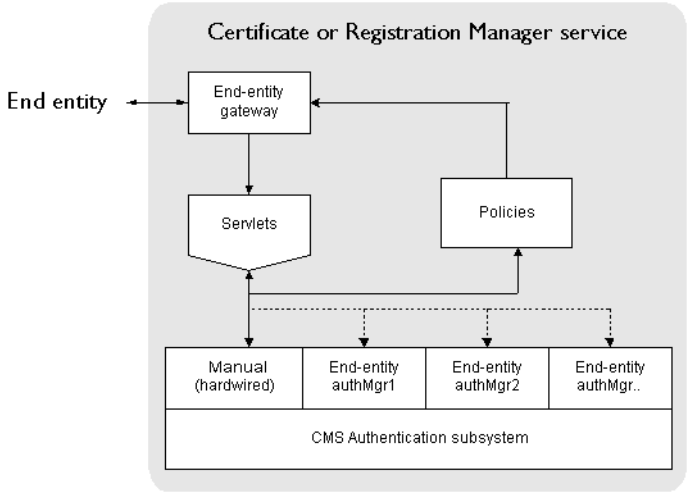


Table 13.1 describes the components shown in Figure 13.1.

Table 13.1 Description of components

Component	Description
End entity	Specifies end entities, such as users, servers, virtual private network (VPN) clients, or routers enrolling for a certificate through a web-based form.
End-entity gateway	Specifies the end-entity gateway provided by Certificate Management System.
Servlets	Specifies end-entity servlets. Each type of request from an end entity is handled by a different servlet; each servlet serves the correct HTML form or other information to the end entity, depending on the protocols supported by that entity.
Policies	Specifies currently configured policies for a Certificate Manager or Registration Manager (depending on which subsystem is processing end-entity requests).

Table 13.1 Description of components (Continued)

Component	Description
Manual	Specifies manual authentication. If you do not configure Certificate Management System to use any of the authentication plug-in modules listed in the Authentication Registration Plugin tab, the server uses manual authentication for end-entity enrollment. This means that all end-entity enrollment requests are queued for agent approval.
End-entity authMgr	Specifies configured authentication managers (based on authentication plug-ins provided out of the box and custom, if any) for authenticating end entities.

How the Architecture Works

The setup shown in Figure 13.1 works in the following way:

1. An end entity enrolls for a certificate through a web-based form that includes fields in which the end entity enters the required enrollment credentials or provides authentication parameter values. The form has been customized to use a specific authentication manager.

2. On the CMS side, a servlet receives the following as input from the form the end entity used:

   - The name of the authentication manager
   - An attribute set (or authentication parameters)

   The servlet then calls the appropriate authentication manager instance to authenticate the end entity.

3. After the end entity is authenticated, policies are applied on the request by either a Certificate Manager or Registration Manager, depending on which subsystem is processing the request.

4. Finally, a certificate is issued and returned to the end entity.

How Authentication Managers Are Used

To authenticate an end entity, an enrollment servlet calls the authentication manager `authenticate` (`IAuthCredentials`) method, passing it credentials received in the HTTP input (the HTML form). The authentication manager implementation specifies which HTTP input variables are required for authentication. For example, in the directory-based authentication manager the required credentials are UID and password. If authentication succeeds, an `AuthToken` is returned. Otherwise, an exception is thrown. `IAuthCredentials` and `AuthToken` contain lists of attribute and value pairs.

The `AuthToken` returned contains the fields and values listed in Table 13.2. The `tokenCertSubject` value must have the subject DN of the certificate set. Values for all other fields are optional.

Table 13.2 AuthToken fields and values

Field	Object type returned
`tokenCertSubject`	`netscape.security.X509.X500Name` (a Java X509Name object)
`tokenCertNotBefore`	`java.util.Date` (a Java Date object)
`tokenCertNotAfter`	`java.util.Date` (a Java Date object)
`tokenCertExts`	`netscape.security.X509.Certificate extensions` (a Java X509Extensions object)

For certificate enrollment the `AuthToken` must contain a field containing a certificate subject name for the authenticated user, for requests made through the `KEYGEN` tag. (Certificate Management System supports certificate requests in `KEYGEN`, `PKCS #10`, `CRMF` and `CEP` formats). The token field is `AuthToken.TOKEN_CERT_SUBJECT`. The value must be an instance of `netscape.security.x509.X500Name`. For other request formats that contain a subject name, a certificate subject name returned in the `AuthToken` will override the subject name in the request, for example in a PKCS #10 request.

In addition, all `AuthTokens` contain the following:

- `authMgrInstName`—the name of the authentication manager (the instance) that authenticated the request
- `authMgrImplName`—the authentication manager's plug-in name (the implementation)
- The time of authentication

Optionally, the `AuthToken` can contain other attributes that customized policies can use in building the certificate.

After authentication, the `AuthToken` and the attributes and values returned in the `AuthToken` are stored in the request. Credentials from the form are discarded. The request is then passed to policies and then to the Certificate Manager for issuance.

Customizing Authentication

Customizing authentication is a five-step process:

- Step 1. Decide on an Authentication Scheme
- Step 2. Write the Authentication Plug-in Module
- Step 3. Register the Authentication Manager Plug-in Module
- Step 4. Create an Instance of the Authentication Plug-in Module
- Step 5. Customize the End-Entity Enrollment Forms

Step 1. Decide on an Authentication Scheme

When you consider customizing authentication, the first thing to decide is the kind of authentication scheme that you want for your end entities when they enroll for a certificate. In planning the scheme, you must identify and decide on

the end-entity attributes Certificate Management System should retrieve from an end-entity request and evaluate the values of. In other words, in this step you must fix the attributes for authentication.

For example, if you have a corporatewide user database, you may want the end entities to provide their user IDs and passwords for the directory as authentication credentials. In that case, you would make sure that the form the end entities use for certificate enrollment includes user ID and password fields. Then end entities will provide this information when they request a certificate from Certificate Management System. Alternatively, you may want to issue certificates to any individual who has a social security number or other kind of valid identification, such as a driver's license or bank account.

Step 2. Write the Authentication Plug-in Module

Authentication managers are implemented as Java classes, which are then registered with Certificate Management System as plug-ins. After you decide on the attributes for authenticating end entities, you need to write an authentication plug-in (Java class) that uses those attributes. Keep in mind that the plug-in implementation must conform to the CMS interface, as explained in the section that follows.

Authentication Manager Plug-in API

To enable you to write custom authentication plug-ins, Certificate Management System provides an authentication manager plug-in API and related classes (Java Docs).

Your authentication manager plug-in must be a Java class that implements the following interface:

```
com.netscape.certsrv.authentication.IAuthManager
```

For the definition of `IAuthManager`, check the CMS SDKs installed at this location: `<server_root>/cms_sdk/sdkdocs`

Compiling and Installing Authentication Manager Plug-ins

When you are compiling an authentication manager plug-in using `javac`, be sure to include CMS classes in the classpath. For example, if a CMS instance named `testCA` is installed in C:`\netscape\server4` (default server root in Windows NT) or `usr/netscape/server4` (default server root in Unix), use the following to compile the authentication manager plug-in implementation:

Windows NT

```
> set CERT40DIR=C:\netscape\server4\bin\cert\jars
> javac -classpath \
C:\jdk1.1.6/lib/classes.zip;%CERT40DIR%/
ldapjdk.jar;%CERT40DIR%/certsrv.jar \
myAuthMgr.java
```

Unix

```
$ set CERT40DIR=usr/netscape/server4/bin/cert/jars
$ javac -classpath \
/usr/jdk1.1.6/lib/classes.zip:$CERT40DIR/
ldapjdk.jar:$CERT40DIR/certsrv.jar \
myAuthMgr.java
```

After compiling an authentication manager plug-in, add it to the CMS authentication framework as explained in the section that follows.

Adding the Authentication Manager Class File to the CMS Authentication Framework

There are two ways in which you can add a custom authentication module class file to the CMS authentication framework. You can either put the class file in the server's default classpath or edit the server's `start-cert` script to include the path to your authentication plug-in module. The recommended method is that you add your class file to the server's default classpath.

The `classes` directories are located here:

`<server_root>/bin/cert/classes` and

`<server_root>/cert-<instance_id>/classes`

- If your authentication plug-in is installation specific—which means, it will be used by all CMS instances under a specific server root—put the class file in this directory: `<server_root>/bin/cert/classes`

- If your authentication manager is instance specific—which means, it will be used by a specific CMS instance—put the class file in this directory: `<server_root>/cert-<instance_id>/classes`

Alternatively, for testing purposes, you can also edit the classpath in the `start-cert` script to point to the custom class file.

Important Keep in mind that the changes you make to the `start-cert` script will take effect only when starting the server from the command line.

The default classpath in the `start-cert` script is as follows:

Windows NT

```
C:\Netscape\Server4\cert-
testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\Netsca
pe\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server4\bin\cert\j
ars\certsrv.jar;C:\Netscape\Server4\java\ldapjdk.jar;C:\Netscap
e\Server4\bin\cert\jre\lib\rt.jar;C:\Netscape\Server4\bin\cert\
jre\lib\i18n.jar;C:\Netscape\Server4\bin\cert\jars\jssjdk12.jar
;C:\Netscape\Server4\java\swingall.jar
```

Unix

```
/usr/netscape/server4/cert-testCA/classes/:/usr/netscape/
server4/bin/cert/classes/:/usr/netscape/server4/bin/cert/jars/
jss.jar:/usr/netscape/server4/bin/cert/jars/certsrv.jar:/usr/
netscape/server4/java/ldapjdk.jar:/usr/netscape/server4/bin/
cert/jre/lib/rt.jar:/usr/netscape/server4/bin/cert/jre/lib/
i18n.jar:/usr/netscape/server4/bin/cert/jars/jssjdk12.jar
```

To add a classpath to the `start-cert` script in Unix:

1. Go the command-line window.
2. Go to the CMS-instance directory. For example, `/usr/netscape/server4/cert-testCA`

3. Enter the following line at the prompt:

   ```
   cat start-cert
   ```

   You should see something similar to this:

   ```
   #!/bin/sh

   /usr/netscape/server4/bin/cert/admin/bin/start -i testCA
   -r /usr/netscape/server4 -e -classpath

   /usr/netscape/server4/bin/cert/classes:/usr/netscape/
   server4/bin/cert/jars/jss.jar:/usr/netscape/server4/bin/
   cert/jars/certsrv.jar:usr/netscape/server4/java/
   ldapjdk.jar:/usr/netscape/server4/bin/cert/jre/lib/
   rt.jar:/usr/netscape/server4/bin/cert/jre/lib/i18n.jar:/
   usr/netscape/server4/bin/cert/jars/jssjdk12.jar
   ```

4. Add your class's directory path to the `start-cert` script. Be sure to add the directory path to the beginning as shown in the example.

   ```
   #!/bin/sh

   /usr/netscape/server4/bin/cert/admin/bin/start -i testCA
   -r /usr/netscape/server4 -e -classpath

   <your_class's_directory_path>:/usr/netscape/server4/cert-
   testCA/classes/:/usr/netscape/server4/bin/cert/classes/:/
   usr/netscape/server4/bin/cert/jars/jss.jar:/usr/netscape/
   server4/bin/cert/jars/certsrv.jar:/usr/netscape/server4/
   java/ldapjdk.jar:/usr/netscape/server4/bin/cert/jre/lib/
   rt.jar:/usr/netscape/server4/bin/cert/jre/lib/i18n.jar:/
   usr/netscape/server4/bin/cert/jars/jssjdk12.jar
   ```

   For example, if your class file is in a directory `/u/jdoe/myAuthMgrs`, here's how the `start-cert` script would look:

   ```
   #!/bin/sh

   /usr/netscape/server4/bin/cert/admin/bin/start -i testCA
   -r /usr/netscape/server4 -e -classpath

   /u/jdoe/myAuthMgrs:/usr/netscape/server4/cert-testCA/
   classes:/usr/netscape/server4/bin/cert/classes/:/usr/
   netscape/server4/bin/cert/jars/jss.jar:/usr/netscape/
   server4/bin/cert/jars/certsrv.jar:/usr/netscape/server4/
   java/ldapjdk.jar:/usr/netscape/server4/bin/cert/jre/lib/
   rt.jar:/usr/netscape/server4/bin/cert/jre/lib/i18n.jar:/
   usr/netscape/server4/bin/cert/jars/jssjdk12.jar
   ```

To add a classpath to the `start-cert.bat` script in Windows NT:

1. Go the command-line window.
2. Go to the CMS instance directory. For example, `C:\netscape\server4\cert-testCA`
3. Enter the following line at the prompt:

   ```
   type start-cert.bat
   ```

   You should see something similar to this:

   ```
   net start cert-testCA /cC:\Netscape\Server4\cert-
   testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\
   Netscape\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server
   4\bin\cert\jars\certsrv.jar;C:\Netscape\Server4\java\ldap
   jdk.jar;C:\Netscape\Server4\bin\cert\jre\lib\rt.jar;C:\Ne
   tscape\Server4\bin\cert\jre\lib\i18n.jar;C:\Netscape\Serv
   er4\bin\cert\jars\jssjdk12.jar;C:\Netscape\Server4\java\s
   wingall.jar
   ```

4. Add your class's directory path to the `start-cert` command. Be sure to add the directory path to the beginning as shown in the example.

   ```
   net start cert-testCA /
   c<your_class's_directory_path>C:\Netscape\Server4\cert-
   testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\
   Netscape\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server
   4\bin\cert\jars\certsrv.jar;C:\Netscape\Server4\java\ldap
   jdk.jar;C:\Netscape\Server4\bin\cert\jre\lib\rt.jar;C:\Ne
   tscape\Server4\bin\cert\jre\lib\i18n.jar;C:\Netscape\Serv
   er4\bin\cert\jars\jssjdk12.jar;C:\Netscape\Server4\java\s
   wingall.jar
   ```

   For example, if your class file is in a directory `C:\jdoe\myAuthMgrs\...` here's how the `start-cert.bat` script would look:

   ```
   net start cert-testCA /
   cC:\jdoe\myAuthMgrs\;C:\Netscape\Server4\cert-
   testCA\classes\;C:\Netscape\Server4\bin\cert\classes\;C:\
   Netscape\Server4\bin\cert\jars\jss.jar;C:\Netscape\Server
   4\bin\cert\jars\certsrv.jar;C:\Netscape\Server4\java\ldap
   jdk.jar;C:\Netscape\Server4\bin\cert\jre\lib\rt.jar;C:\Ne
   tscape\Server4\bin\cert\jre\lib\i18n.jar;C:\Netscape\Serv
   er4\bin\cert\jars\jssjdk12.jar;C:\Netscape\Server4\java\s
   wingall.jar
   ```

Authentication Manager Examples

To aid you in writing custom authentication managers, Certificate Management System provides sample authentication plug-ins. You can find them installed at this location:

```
<server_root>/cms_sdk/samples/authentication
```

Step 3. Register the Authentication Manager Plug-in Module

You can register custom authentication manager plug-in modules by using the CMS window. Before registering a custom plug-in, be sure to put the Java class for the plug-in in the `classes` directory. For instructions, see "Registering an Authentication Module" on page 414.

Step 4. Create an Instance of the Authentication Plug-in Module

After you have registered the custom authentication plug-in module, you must configure an instance of it. For instructions, see "Step 4: Add an Authentication Instance" on page 395.

Step 5. Customize the End-Entity Enrollment Forms

After you have created the authentication manager instance, you must customize the appropriate HTML form for end-entity enrollment. Make the following changes:

- Update the enrollment forms to include all the required attributes.
- Update the enrollment forms to use the new authentication manager instance. This involves setting the value of `authenticator`, an HTML element in the enrollment form, to the name of the custom authentication manager instance.

The most convenient way to do this is by editing the forms provided out of the box for end entities. For example, you can add new fields to the directory-based authentication form provided for end-entity enrollment. You can also use custom enrollment forms, but if you do so, be sure to include the following HTML element in your end-entity enrollment form:

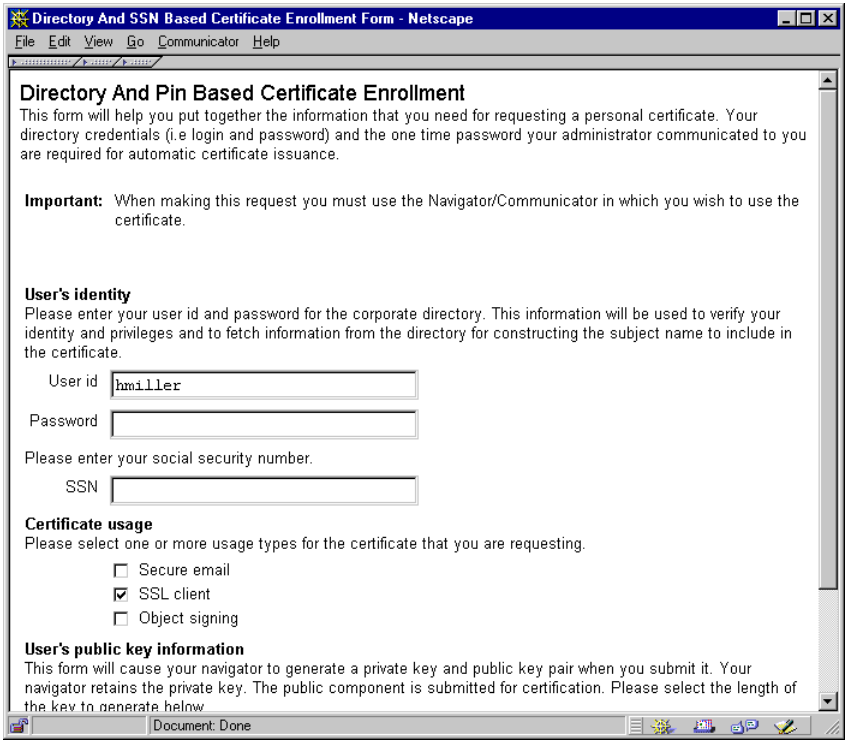
```
<INPUT TYPE="HIDDEN" NAME="authenticator" VALUE="myAuthMgr">
```

where `myAuthMgr` is the name of the authentication instance you want to use with the enrollment form.

For more information, see "Step 5. Set Up the Enrollment Interface" on page 400.

Figure 13.2 shows a directory-based authentication form customized to use a social security number (SSN) for authentication in addition to user ID and password for the authentication directory. Note the text field named *SSN*.

Figure 13.2 End-entity enrollment form for SSN and directory-based authentication



5

Job Scheduling and Notification

Chapter

14

Introduction to Job Scheduling and Notifications

Netscape Certificate Management System (CMS) includes a component called *Job Scheduler* that can execute specific jobs at specified times. The job scheduler functions similar to a traditional Unix *cron* daemon in that it takes registered cron jobs and executes them at a preconfigured date and time. If configured, the scheduler checks at specified intervals for jobs waiting to be executed; if the specified execution time has arrived, the scheduler initiates the job automatically. Jobs that you might want to schedule include email notifications of timed events (such as the expiration of a certificate) that require action on the part of users, and periodic activities such as updates of related directories.

You can also configure Certificate Management System to send email notifications automatically to end entities, agents, or administrators when certain events occur. Unlike jobs that are executed on a preconfigured schedule, these notifications are event-driven—that is, whenever an event occurs, the server notifies the user. Notifiable events include certificate issuance and pending requests in an agent queue.

This chapter describes the job plug-in modules and event notifications that come with Certificate Management System and explains how to schedule times for jobs.

The chapter has the following sections:

Overview of Job Plug-in Modules

Both the Certificate Manager and Registration Manager provide a set of job plug-in modules that can be employed by the server to automate certain activities. Table 14.1 lists these modules.

Jobs are implemented as Java classes, which are then registered with Certificate Management System as plug-in modules. You can use a given implementation of a job module and configure multiple instances of it. Each instance must have a unique name (an alphanumeric string with no spaces) and can contain different input parameter values to apply to different jobs. In other words, a given job implementation can be shared by multiple configurations.

Table 14.1 Schedulable job plug-in modules for Certificate Manager and Registration Manager

Plug-in module name	Description
`RenewalNotificationJob`	A schedulable job that notifies end entities by email that their certificates are about to expire and must be renewed, and optionally sends a summary of these notices to agents. For more information, see "Certificate Renewal Notifications" on page 435.
`RequestInQueueJob`	A schedulable job that notifies agents at regular intervals of the current state of the request queue. In addition, agents can also be notified by email that a request has been added to the request queue by configuring an event-driven notification. See "Notification of Request Queue Status" on page 440.
`UnpublishExpiredJob`	A schedulable job that updates the configured publishing directory periodically by removing expired certificates, and sends a summary of removed certificates to agents or administrators. For more information, see "Directory Update and Notification" on page 444.

Certificate Renewal Notifications

When a certificate is about to expire, the owner of the certificate needs to renew it. Using the Jobs Scheduler, you can configure a Certificate Manager or Registration Manager to automatically send email-based renewal notices to users whose certificates are about to expire or have expired. You can also configure these subsystems to send one or more administrators or issuing agents a summary of users who have received these reminders.

RenewalNotificationJob Module

The java class that implements the job for scheduling renewal notifications is identified as follows:

```
com.netscape.certsrv.jobs.RenewalNotificationJob
```

- In the CMS window, the module is identified as follows: `RenewalNotificationJob`
- In the CMS configuration file, the module is identified as follows: `jobsScheduler.impl.RenewalNotificationJob.class=com.netscape.certsrv.jobs.RenewalNotificationJob`

This plug-in is a schedulable job. When an instance of the job is enabled, it checks for certificates that are about to expire in the internal database. When it finds one, it automatically emails the certificate's owner and continues sending email reminders for a configured period of time, or until the certificate is renewed. The job also collects a summary of all such renewal notifications and mails the summary to one or more agents or administrators.

The job determines the email address to which to send the notification using an email resolver, which you can customize. By default, the email address is found in the certificate itself or in the certificate's associated enrollment request.

The email notification message, as well as the summary message, are constructed using a template found in the configured directory. This directory has the following default location:

```
<server_root>/cert-<instance_id>/emails
```

You can configure both the path and filenames of the template files for each job and modify the templates to customize the contents and appearance of the messages. Messages can be sent as HTML or plain text. For customization information, see "Customizing Notification Messages" on page 455.

For each instance of the `RenewalNotificationJob` class, you can configure the following:

- The schedule of times when the job will be run; see "Schedule for Executing Jobs" on page 448.
- How long before expiration the first notification will be sent.
- How long, after the certificate expires, notifications will continue to be sent if the certificate is not renewed.
- The sender of the notification messages (who will be notified of any delivery problems).
- The file location of the notification email template.
- The subject line of the notification message.
- How the email address for the notification is to be resolved.
- Whether a summary will be compiled and sent.

If a summary is to be sent, you can configure the following:

- The recipients of the summary message. These can be, for example, agents who need to know the status of user certificates.
- The sender of the summary message (who will be notified of any delivery problems).
- The file location of the summary message template.
- The file location of content and format of each item to be collected for the summary.
- The subject line of the summary message.

Figure 14.1 shows how the configurable parameters pertaining to the `RenewalNotificationJob` plug-in module are displayed in the CMS window.

Figure 14.1 Parameters defined in the RenewalNotificationJob module

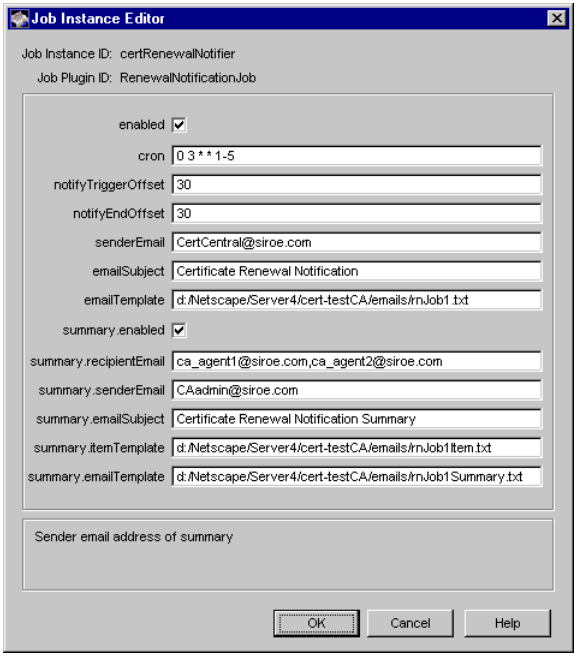


Table 14.2 gives details about each of these parameters.

Table 14.2 Description of parameters defined in the RenewalNotificationJob module

Parameter	Description
`enabled`	Specifies whether the job is enabled or disabled. Check the box to enable the job. Uncheck the box to disable the job. If you enable the job and set the remaining parameters correctly, the server runs the job at scheduled intervals.
`cron`	Specifies the `cron` specification for when this job should be run. In other words, it specifies the time at which the Job Scheduler daemon thread should check the certificates for sending renewal notifications. Permissible values: Must follow the convention specified in "Schedule for Executing Jobs" on page 448. Example: `03**1-5`

Table 14.2 Description of parameters defined in the RenewalNotificationJob module (Continued)

Parameter	Description
`notifyTriggerOffset`	Specifies how long (in days) before certificate expiration the first notification will be sent. Permissible values: As applicable. Example: `30`
`notifyEndOffset`	Specifies how long (in days) after the certificate expire notifications will continue to be sent, if the certificate is not renewed. Permissible values: As applicable. Example: `30`
`senderEmail`	Specifies the sender of the notification messages (who will be notified of any delivery problems). Permissible values: The complete email address. Example: `CertCentral@siroe.com`
`emailSubject`	Specifies the subject line of the notification message. Permissible values: An alphanumeric string of up to 255 characters. Example: `Certificate Renewal Notification`
`emailTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the message content. Permissible values: Template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/renewJob.txt`
`summary.enabled`	Specifies whether a summary report of renewal notifications should be compiled and sent. Check the box if you want the server to compose and send a summary report. Uncheck the box if you don't want the server to compose and send a summary report. If you check the box, be sure to set the remaining parameters; these are required by the server to send the summary report.

Table 14.2 Description of parameters defined in the RenewalNotificationJob module (Continued)

Parameter	Description
`summary.recipientEmail`	Specifies the recipients of the summary message. These can be, for example, agents who need to know the status of user certificates. Permissible values: Full email addresses, separated by commas. Example: `ca_agent1@siroe.com,ca_agent2@siroe.com`
`summary.senderEmail`	Specifies the sender of the summary message (who will be notified of any delivery problems). Permissible values: The full email address. Example: `CAadmin@siroe.com`
`summary.emailSubject`	Specifies the subject line of the summary message. Permissible values: An alphanumeric string of up to 255 characters. Example: `Certificate Renewal Notification Summary`
`summary.itemTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the content and format of each item to be collected for the summary report (see the `summary.emailTemplate` parameter below). For details, see "Customizing Notification Messages" on page 455. Permissible values: The template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/renewJobItem.txt`
`summary.emailTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the summary report. For details, see "Customizing Notification Messages" on page 455. Permissible values: The template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/renewJobSummary.txt`

Notification of Request Queue Status

In addition to or instead of notifying agents of new requests, you might want to schedule a job that regularly notifies them of the status of the request queue. Such a job can check at a configured interval whether there are any *deferred* enrollment requests waiting for review. It can then send an email message to agents informing them of the number of requests waiting in the request queue for which they are responsible.

RequestInQJob Module

The java class that implements the job for request-queue-status notifications is identified as follows:

```
com.netscape.certsrv.jobs.RequestInQJob
```

- In the CMS window, the module is identified as follows: `RequestInQJob`
- In the CMS configuration file, the module is identified as follows: `jobsScheduler.impl.RequestInQJob.class=com.netscape.certsrv.jobs.RequestInQJob`

This plug-in is a schedulable job. When an instance of the job is enabled, it gets activated at the configured interval and checks the status of the request queue. If any deferred enrollment requests are waiting in the queue, the job constructs an email message summarizing its findings and sends it to the specified agents.

The job constructs the summary message by using a template located in a configured directory. This directory has the following default location:

```
<server_root>/cert-<instance_id>/emails
```

You can configure the path and filename of the template file for each job and modify the templates to customize the contents and appearance of the messages. Messages can be sent as HTML or plain text.

For each instance of the `RequestInQJob` class, you can configure the following:

- The subsystem, Certificate Manager or Registration Manager, that should use this job.
- The schedule of times when the job will be run; see "Schedule for Executing Jobs" on page 448.

- The sender of the notification messages (who will be notified of any delivery problems).
- The file location of the notification email template.
- The subject line of the notification message.
- The email addresses of message recipients; these should be subsystem agents whose task it is to review manual enrollment requests.

Figure 14.2 shows how the configurable parameters for the `RequestInQJob` module are displayed in the CMS window.

Figure 14.2 Parameters defined in the RequestInQJob module

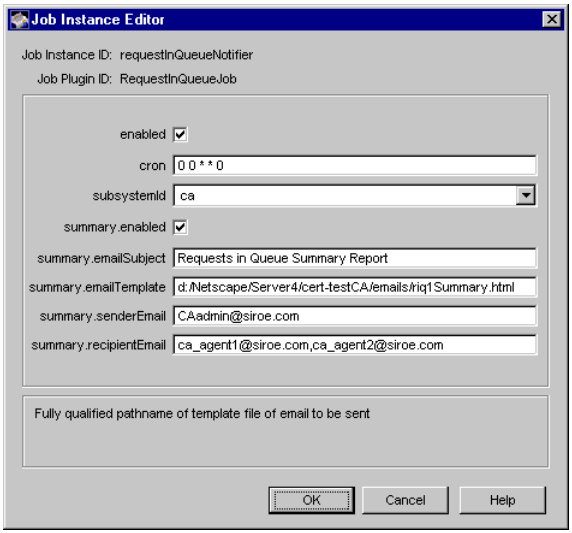


Table 14.3 gives details for each of these parameters.

Table 14.3 Description of parameters defined in the RequestInQJob module

Parameter	Description
`enabled`	Specifies whether the job is enabled or disabled. Check the box to enable the job. Uncheck the box to disable the job. If you enable the job and set the remaining parameters correctly, the server runs the job at scheduled intervals.

Table 14.3 Description of parameters defined in the RequestInQJob module (Continued)

Parameter	Description
`cron`	Specifies the `cron` specification for when this job should be run. This is the time at which the Job Scheduler daemon thread checks the queue for pending requests. Permissible values: Must follow the convention specified in "Schedule for Executing Jobs" on page 448. Example: `00**0`
`subsystemid`	Specifies the subsystem that this job is for. Permissible values: `ca` or `ra`. • `ca` specifies that the job is for the Certificate Manager. • `ra` specifies that the job is for the Registration Manager. Example: `ca`
`summary.enabled`	Specifies whether a summary of the job accomplished should be compiled and sent. Check the box if you want the server to compose and send a summary report. Uncheck the box if you don't want the server to compose and send a summary report. If you check the box, be sure to set the remaining parameters; these are required by the server to send the summary report.
`summary.emailSubject`	Specifies the subject line of the summary message. Permissible values: An alphanumeric string of up to 255 characters. Example: `Summary Report of Requests in the Agent Queue`
`summary.emailTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the summary report. For details, see "Customizing Notification Messages" on page 455. Permissible values: The template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/reqInQJobSummary.txt`

Table 14.3 Description of parameters defined in the RequestInQJob module (Continued)

Parameter	Description
`summary.senderEmail`	Specifies the sender of the notification message (who should be notified of any delivery problems). Permissible values: The full email address. Example: `CAadmin@siroe.com`
`summary.recipientEmail`	Specifies the recipients of the summary message. These should be, for example, agents who need to process pending requests. Permissible values: Full email addresses, separated by commas. Example: `ca_agent1@siroe.com,ca_agent2@siroe.com`

Directory Update and Notification

Certificate Management System doesn't automatically remove expired certificates from the publishing directory. If you configure a Certificate Manager or Registration Manager to publish certificates to an LDAP directory, over time the directory will contain expired certificates. To help you remove expired certificates from the directory, both the Certificate Manager and Registration Manager come with a plug-in module that allows you to create a schedulable job that periodically removes (or unpublishes) certificates that have expired. When the directory has been updated, the job can collect a summary report of the certificates that have been removed and send it to people who need to have this information. Typically, you would want to send this notification to certificate issuing agents or the administrator of the publishing directory.

Note that the job automates removal of expired certificates from the directory. You can also remove expired certificates manually following the instructions in "Manually Updating Certificates in the Directory" on page 838.

UnpublishExpiredJob Module

The java class that implements the job for scheduling removal of expired certificates from the directory is identified as follows:

`com.netscape.certsrv.jobs.UnpublishExpiredJob`

- In the CMS window, the module is identified as follows:
  `UnpublishExpiredJob`
- In the CMS configuration file, the module is identified as follows:
  `jobsScheduler.impl.ExpiredUnpublishJob.class=com.netscape.certsrv.jobs.UnpublishExpiredJob`

This plug-in is a schedulable job. When an instance of the job is enabled, it gets activated at the configured interval and checks for certificates that have expired and are still marked as *published* in the internal database. The job connects to the publishing directory and deletes these certificates; it then marks these certificates as *unpublished* in the internal database. The job also collects a summary of expired certificates that it deleted and mails the summary to one or more agents or administrators as specified by the configuration.

The job constructs the summary message by using a template located in a configured directory. This directory has the following default location:

`<server_root>/cert-<instance_id>/emails`

You can configure the path and filename of the template file for each job. You can also modify the templates to customize the contents and appearance of the messages; see "Customizing Message Templates" on page 459.

Messages can be sent as HTML or plain text.

For each instance of the `UnpublishExpiredJob` class, you can configure the following:

- The schedule of times when the job will be run; see "Schedule for Executing Jobs" on page 448.
- Whether a summary will be compiled and sent.

If a summary is to be sent, you can configure the following:

- The recipients of the summary message. These can be, for example, administrators who are responsible for the publishing directory.
- The sender of the summary message (who will be notified of any delivery problems).
- The file location a of the summary message template.

- The file location of content and format of each item to be collected for the summary.
- The subject line of the summary message.

Figure 14.3 shows how the configurable parameters for the `UnpublishExpiredJob` module are displayed in the CMS window.

Figure 14.3 Parameters defined in the UnpublishExpiredJob module

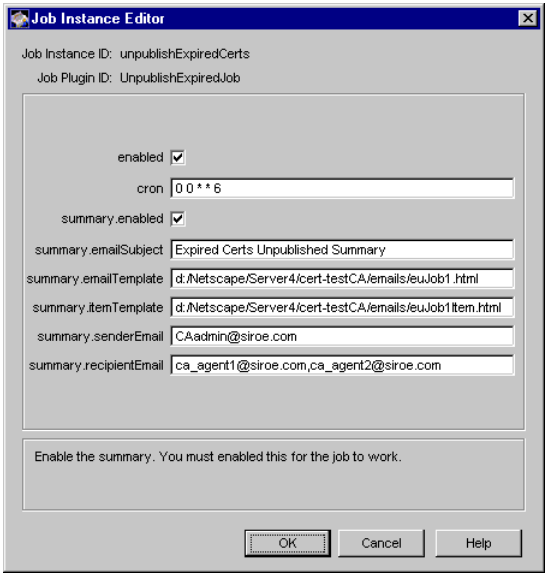


Table 14.4 gives details for each of these parameters.

Table 14.4 Description of parameters defined in the UnpublishExpiredJob module

Parameter	Description
`enabled`	Specifies whether the job is enabled or disabled. Check the box to enable the job. Uncheck the box to disable the job. If you enable the job and set the remaining parameters correctly, the server runs the job at scheduled intervals.

Table 14.4 Description of parameters defined in the UnpublishExpiredJob module (Continued)

Parameter	Description
`cron`	Specifies the `cron` specification for when this job should be run. This is the time at which the Job Scheduler daemon thread checks the certificates for removing expired certificates from the publishing directory. Permissible values: Must follow the convention specified in "Schedule for Executing Jobs" on page 448. Example: `00**6`
`summary.enabled`	Specifies whether a summary of the certificates removed by the job should be compiled and sent. Check the box if you want the server to compose and send a summary report. Uncheck the box if you don't want the server to compose and send a summary report. If you check the box, be sure to set the remaining parameters; these are required by the server to send the summary report.
`summary.emailSubject`	Specifies the subject line of the summary message. Permissible values: An alphanumeric string of up to 255 characters. Example: `Expired Certificate Removal Job Summary`
`summary.emailTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the summary report. For details, see "Customizing Notification Messages" on page 455. Permissible values: The template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/unpublishCertsJobSummary.html`
`summary.itemTemplate`	Specifies the path, including the filename, to the directory that contains the template to be used for formulating the content and format of each item to be collected for the summary report (see the `summary.emailTemplate` parameter above). Permissible values: The template file path, including the file name. Example: `C:/Netscape/Server4/cert-testCA/emails/unpublishCertsJobItem.txt`

Table 14.4 Description of parameters defined in the UnpublishExpiredJob module (Continued)

Parameter	Description
`summary.senderEmail`	Specifies the sender of the summary message (who should be notified of any delivery problems). Permissible values: The full email address. Example: `CAadmin@siroe.com`
`summary.recipientEmail`	Specifies the recipients of the summary message. These can be, for example, agents who need to know the status of user certificates. Permissible values: Complete email addresses, separated by commas. Example: `cert_agent1@siroe.com,cert_agent2@siroe.com`

Schedule for Executing Jobs

The Job Scheduler uses a variation of the Unix `crontab` entry format to specify dates and times for checking the job queue and executing jobs. As shown in Table 14.5, the time entry format consists of five fields (the sixth field specified for the Unix `crontab` is not used by the Job Scheduler). Values are separated by spaces or tabs.

Table 14.5 Time format for scheduling jobs

Field	Value
Minute	0-59
Hour	0-23
Day of month	1-31
Month of year	1-12
Day of week	0-6 (where 0=Sunday)

Each field can contain either a single integer or a pair of integers separated by a hyphen or dash (`-`) to indicate an inclusive range. To specify all legal values, a field can contain an asterisk rather than an integer. Day fields can contain a comma-separated list of values.

For example, the following time entry specifies every hour at 15 minutes (1:15, 2:15, 3:15 and so on):

```
15 * * * *
```

The following example specifies a job execution time of noon on April 12:

```
0 12 12 4 *
```

The day-of-month and day-of-week fields can contain a comma-separated list of values to specify more than one day. If both day fields are specified, the specification is inclusive; that is, the day of the month is not required to fall on the day of the week to be valid. For example, the following entry specifies a job execution time of midnight on the first and fifteenth of every month, *and* on every Monday:

```
0 0 1,15 * 1
```

To specify one day type without the other, use an asterisk in the other day field. For example, the following entry specifies a job execution time of 3:15 a.m. on every weekday morning:

```
15 3 * * 1-5
```

Event-Driven Notifications

You can configure a Certificate Manager or Registration Manager to send email notifications automatically to end entities, agents, or administrators when certain events occur. Unlike jobs that are executed at a preconfigured schedule by the Job Scheduler component, these notifications are event driven—that is, whenever the specified event occurs, the server notifies the configured user.

Notifiable events include the following:

- Notifications of Certificate Issuance to End Entities

  End entities are notified by email that a requested certificate has been issued. This is an event-driven notification.

- Notification of New Request in Queue

Agents are notified by email that a request has been added to the request queue. Alternatively (or in addition) a schedulable job can notify agents at regular intervals of the current state of the request queue; see "Notification of Request Queue Status" on page 440.

Notifications of Certificate Issuance to End Entities

You can configure the Certificate Manager or Registration Manager to send a notification message to users who have been issued certificates in response to enrollment requests. This message normally includes information about the issued certificate and instructions for importing the certificate into the user's client.

This kind of notification involves a listener class in the subsystem that registers an interest in an appropriate event, in this case successful completion of an enrollment request. In the CMS configuration, this listener class for a Certificate Manager is defined as `ca.notification.certIssued` and for the Registration Manager it is defined as `ra.notification.certIssued`.

For more information on listeners, check the samples directory:

```
<server_root>/cms_sdk/samples/listeners
```

When a certificate is issued, the listener builds a notification message based on a configured template and sends it to an email address that it determines by using an email resolver. By default the email is obtained from the email address entered in the request or from the certificate.

- The email resolver first checks the request for the email address and if doesn't find one, it checks the subject name of the certificate for the email address; if the subject name doesn't include the email address, the resolver checks the certificate for the Subject Alternative Name extension to see whether it specifies the email address. For specifying an email address in the Subject Alternative Name extension, see "Subject Alternative Name Extension Policy" on page 668.

- In the absence of an email address, the notification is sent to the email address specified in the "Sender's Email Address" field, instead of the requestor, as an undeliverable notification. There'll also be a message to this effect in the logs; see "Monitoring Logs" on page 1064.

Note that you can customize the email resolver using the `ReqCertSANameEmailResolver.java` class included as a sample at this location: `<server_root>/cms_sdk/samples/resolvers`

The template that the listener uses to construct the email notification message is located in the configured directory. This directory has the following default location: `<server_root>/cert-<instance_id>/emails`

You can configure both the path and filename of the template file. You can also modify the template to customize the contents and appearance of the messages; see "Customizing Message Templates" on page 459.

Messages can be sent as HTML or plain text.

For the `certIssued` listener, you can configure the following:

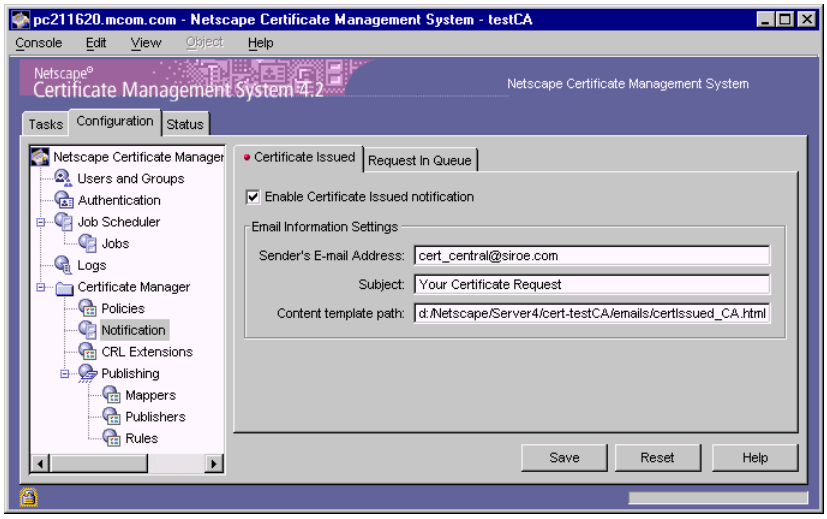
- Whether the listener is enabled.
- The sender of the notification messages (who will be notified of any delivery problems).
- The location of the notification email template.
- The subject line of the notification message.

Configuring a Subsystem to Send Notifications to End Entities

To configure a Certificate Manager or Registration Manager to send certificate-issuance notifications to end entities:

1. Access the CMS window (see "Logging In to the CMS Window" on page 78).
2. Click the Configuration tab.
3. In the navigation tree, select the subsystem, then click Notification. (The figure below shows the Certificate Manager's notification feature; the Registration Manager also has a similar feature.)

The Certificate Issued tab appears.



4. To enable the notification feature, check the "Enable Certificate Issued notification" option.

5. In the Email Information Settings section, enter information as appropriate:

   Sender's E-mail Address. Type the sender's full email address (this is the person who should be notified of any delivery problems).

   Subject. Type the subject title for the notification.

   Content template path. Type the path, including the filename, to the directory that contains the template to be used for formulating the message content.

6. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Notification of New Request in Queue

When a *deferred* end-entity request enters the request queue of a Certificate Manager or Registration Manager, agents assigned to manage the queue must review the request and reject or accept it. To help ensure that an agent processes the request in a timely manner, you can configure the subsystem to notify agents whenever a new request gets added to the request queue.

This kind of notification involves a *listener* class in the subsystem that registers an interest in an appropriate event, in this case the addition of a request to the request queue. In the CMS configuration, this listener class is identified as follows: `ca.notification.requestInQ`

For more information on listener classes, check the samples directory:

`<server_root>/cms_sdk/samples/listeners`

When a request is added to the queue, the listener builds a notification message based on a configured template and sends it to one or more agents' email addresses as configured.

The template that the listener uses to construct the email notification message is located in the configured directory. This directory has the following default location:

`<server_root>/cert-<instance_id>/emails`

You can configure both the path and filename of the template file. You can also modify the template to customize the contents and appearance of the messages; see "Customizing Message Templates" on page 459.

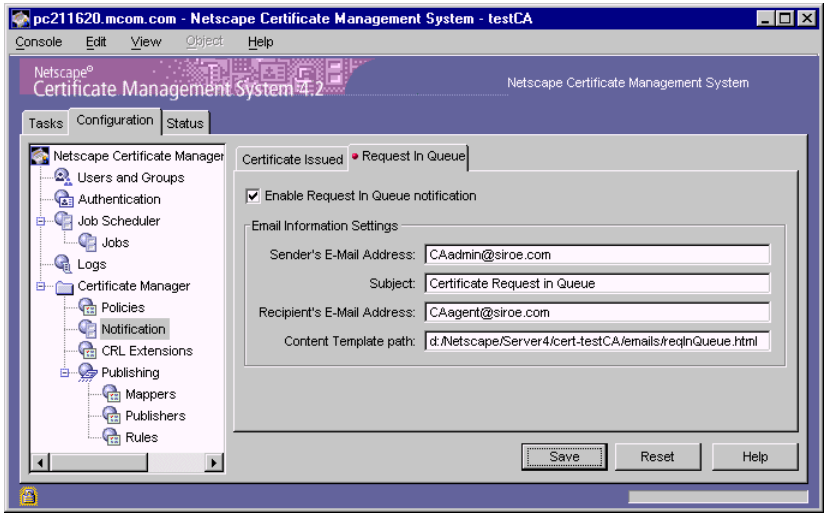
For the `requestInQ` event listener, you can configure the following:

- Whether the listener is enabled.
- The sender of the notification messages (who will be notified of any delivery problems).
- The location of the notification email template.
- The subject line of the notification message.
- The email addresses of message recipients; these should be subsystem agents whose task is to review *deferred* enrollment requests.

Configuring a Subsystem to Send Request Queue Notifications

To configure a Certificate Manager or Registration Manager to send email notifications to its agents:

1. Access the CMS window (see “Logging In to the CMS Window” on page 78).
2. Click the Configuration tab.
3. In the navigation tree, select the subsystem, and then click Notification. (The figure below shows the Certificate Manager’s notification feature; the Registration Manager also has a similar feature.)
4. In the right pane, click Request In Queue.



5. To enable the notification feature, check the “Enable Request In Queue notification” option.
6. Enter information as appropriate:

   Sender's E-Mail Address. Type the sender’s full email address (this is the person who should be notified of any delivery problems).

Subject. Type the subject title for the notification—for example, "End Entity Request in Queue."

Recipient's E-Mail Address. Type the recipient's full email address (this is the person who will check the queue). You can specify more than one recipient; separate email addresses by commas.

Content template path. Type the path, including the filename, to the directory that contains the template to be used for formulating the message content.

7. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Customizing Notification Messages

Notification and summary email messages are constructed using templates located in the `emails` directory of a CMS instance. This directory has the following default location:

```
<server_root>/cert-<instance_id>/emails
```

Both text an HTML templates are included by default. They are listed in Table 14.6 and Table 14.7.

Templates for Event-Triggered Notifications

Table 14.6 lists the default template files provided for formulating event-triggered-notification messages. You can customize certain aspects of these templates—such as the subject of the email message and the location and name of the template file—using the CMS window.

Table 14.6 Default templates for event-triggered notifications

Filename	Description
`certIssued_CA`	Template for the Certificate Manager to send plain-text notifications to end entities upon issuance of certificates.
`certIssued_CA.html`	Template for the Certificate Manager to send HTML-based notifications to end entities upon issuance of certificates.
`certIssued_RA`	Template for the Registration Manager to send plain-text notifications to end entities upon issuance of certificates.
`certIssued_RA.html`	Template for the Registration Manager to send HTML-based notifications to end entities upon issuance of certificates.
`certRequestRejected_CA`	Template for the Certificate Manager to send plain-text notifications to end entities when rejecting their certificate requests.
`certRequestRejected_CA.html`	Template for the Certificate Manager to send HTML-based notifications to end entities when rejecting their certificate requests.
`certRequestRejected_RA`	Template for the Registration Manager to send plain-text notifications to end entities when rejecting their certificate requests.
`certRequestRejected_RA.html`	Template for the Registration Manager to send HTML-based notifications to end entities when rejecting their certificate requests.
`reqInQueue`	Template for the Certificate Manager or Registration Manager to send plain-text notifications to agents when a request enters the queue.
`reqInQueue.html`	Template for the Certificate Manager or Registration Manager to send plain-text notifications to agents when a request enters the queue.

Note that the email notification that a certificate has been issued is based on a template file whose default name begins with `certIssued`. Similarly, the email notification that a certificate has been rejected is based on a template file whose name begins with `certRequestRejected`. This template file must be located in the same directory as the certificate-issuance template. Unlike the certificate-issuance template, the filename of the certificate-rejection template (`certRequestRejected`) cannot be changed. However, the file extension for

the `certRequestRejected` file can be changed, as long as it exactly matches the file extension specified for the certificate issuance template file. For example, if the certificate issuance template file is named `certIssued_CA.htm`, the `certRequestRejected` file must be named `certRequestRejected.htm`. The HTML file extensions permitted are `.htm`, `.html`, `.HTM`, and `.HTML`. Template files with any other extension (or no extension) are treated as text files.

If you change the name of any of these files, be sure to make the appropriate changes to the configuration (see the "Content template file" field on page 452 and page 455). In the CMS configuration, template files for event-triggered notifications are identified as follows:

```
<subsystem>.notification.<notification_name>.emailTemplate=
<template_file_path>
```

```
<subsystem>.notification.<notification_name>.emailTemplate=
<template_file_path>
```

`<subsystem>` specifies the prefix that identifies the subsystem to which the configuration parameter belongs—`ca` for the Certificate Manager and `ra` for the Registration Manager.

`<notification_name>` specifies the name of the event-triggered notification—`certIssued` for the certificate issuance notifications to end entities and `requestInQ` for the request in queue notifications to agents.

`<template_file_path>` specifies the path, including the filename, to the directory that contains the template to be used for formulating the message content.

Tokens, which you can use as variables in the body of the message, are also defined for each template, enabling you to customize the message; a token is replaced by its current variable value in the constructed message. For details, see "Customizing Message Templates" on page 459.

Templates for Summary Notifications

Table 14.7 lists the default template files for formulating the notification messages that summarize jobs that were executed by the Job Scheduler component of a Certificate Manager or Registration Manager. You can change the name of these files as applicable; be sure to make the appropriate changes to the configuration.

For summaries, a separate template is used to format the entry for each item in the summary. The item entries are then added to a table in the summary message.

Tokens, which you can use as variables in the body of the message, are defined for each templates enabling you to customize the message; the token is replaced by its current variable value in the constructed message. For details, see "Customizing Message Templates" on page 459.

Table 14.7 Default templates for summary notifications

Filename	Description
Templates for `UnpublishExpiredJob` module	
`ExpiredUnpublishJob`	Template for formulating the summary report or table that summarizes removal of expired certificates from the directory.
`ExpiredUnpublishJobItem`	Template for formatting the items to be included in the summary table, which is constructed using the `ExpiredUnpublishJob` template.
Templates for `RequestInQueueJob` module	
`riq1Item.html`	Template for formatting the items to be included in the summary table, which is constructed using the `riq1Summary.html` template.
`riq1Summary.html`	Template for formulating the summary report or table that summarizes how many requests are pending in the agent queue of a Certificate Manager or Registration Manager.
Templates for `RenewalNotificationJob` module	
`rnJob1.txt`	Template for formulating the message content to be sent to end entities to inform them that their certificates are about to expire and that they should renew their certificates before expiration.
`rnJob1Item.txt`	Template for formatting the items to be included in the summary report, which is constructed using the `rnJob1Summary.txt` template.
`rnJob1Summary.txt`	Template for formulating the summary report to be sent to agents and administrators.

Note that in the CMS configuration, template files for schedulable jobs are identified as follows:

```
jobsScheduler.job.<job_name>.summary.emailTemplate=
<template_file_path>
```

```
jobsScheduler.job.<job_name>.summary.itemTemplate=
<template_file_path>
```

`<job_name>` specifies the job instance name. For a list of job instances created by default, see Table 15.1 on page 473.

`<template_file_path>` specifies the path, including the filename, to the directory that contains the template to be used for formulating the message content.

Customizing Message Templates

You can modify the templates to customize the contents and appearance of messages. The message body can contain HTML or plain text. In the body of the message, you can use tokens or keywords as variables. A token is indicated by the dollar character (`$`) and is replaced by its current variable value in the constructed message. Different tokens are available for each job or notification class. These are listed in "Tokens Available in Message Templates" on page 460.

For example, a certificate-issuance-notification message can make use of tokens as follows:

CERTIFICATE ISSUANCE NOTIFICATION

Your certificate request (`$RequestId`) has been processed successfully. Details of your certificate are as follows:

Serial Number= `$SerialNumber`

SubjectDN= `$SubjectDN`

IssuerDN= `$IssuerDN`

Validity Period= `$NotBefore` - `$NotAfter`

To get your certificate, please follow this URL:

https://`$HttpHost`:`$HttpPort`/getCertFromRequest?requestId=`$RequestId`

If you have any questions or problems, please send an email to cert_central@siroe.com.

Thank you.

Tokens Available in Message Templates

This section explains the tokens provided in the templates used by the default job plug-in and event-triggered notification modules to formulate notification messages.

- Tokens for Certificate Issuance Notifications to End Entities
- Tokens for Rejection Notifications to End Entities
- Tokens for Renewal Notification Messages
- Tokens for Request In Queue Notification Messages
- Tokens for Directory Update Notification Messages

Tokens for Certificate Issuance Notifications to End Entities

Table 14.8 lists the tokens that are available in the message templates provided for formulating the content of email notifications to end entities; a Certificate Manager or Registration Manager can send these notifications upon issuance of the certificates they requested.

Table 14.8 Tokens defined in templates used for certificate-issuance notifications

Token	Description
`$HttpHost`	Specifies the fully qualified host name of the Certificate Manager or Registration Manager to which end entities should connect to retrieve their certificates. (This token enables you to construct the URL from which end entities can download their certificates; see the example in "Customizing Message Templates" on page 459.)
`$HttpPort`	Specifies the port number at which the Certificate Manager or Registration Manager is listening to end-entity requests. (This token enables you to construct the URL from which end entities can download their certificates; see the example in "Customizing Message Templates" on page 459.)
`$InstanceID`	Specifies the ID assigned to the subsystem that sent this notification. • If the notification is sent by a Certificate Manager, this will be `ca`. • If the notification is sent by a Registration Manager, this will be `ra`.
`$IssuerDN`	Specifies the distinguished name of the CA that issued the certificate.

Table 14.8 Tokens defined in templates used for certificate-issuance notifications (Continued)

Token	Description
`$NotAfter`	Specifies the `NotAfter` attribute.
`$NotBefore`	Specifies the `NotBefore` attribute.
`$RecipientEmail`	Specifies the email address of the recipient (the address resolved from the email resolver explained in "Notifications of Certificate Issuance to End Entities" on page 450).
`$RequestId`	Specifies the request ID.
`$SerialNumber`	Specifies the serial number of the certificate that has been issued.
`$SenderEmail`	Specifies the email address of the sender (it is the same as the one you specify in the `Sender's E-mail Address` field in "Configuring a Subsystem to Send Notifications to End Entities" on page 451).
`$SubjectDN`	Specifies the distinguished name of the certificate subject.

Tokens for Rejection Notifications to End Entities

Table 14.9 lists tokens that are available in the message templates provided for formulating the content of email notifications to end entities; a Certificate Manager or Registration Manager can send these notifications to end entities when rejecting certificate requests.

Table 14.9 Tokens defined in templates used for request-rejection notifications

Token	Description
`$InstanceID`	Specifies the ID assigned to the subsystem that sent this notification. • If the notification is sent by a Certificate Manager, this will be `ca`. • If the notification is sent by a Registration Manager, this will be `ra`.
`$RequestId`	Specifies the request ID.

Tokens for Renewal Notification Messages

This section lists the tokens that are available in the message templates for instances of the `RenewalNotificationJob` class or plug-in module.

Table 14.10 lists the tokens that you can use for formulating this job's summary report. You can customize the content and format of the items in the report by using the tokens defined in Table 14.11.

Table 14.10 Tokens for the renewal-notification job's summary report

Token	Description
`$ExecutionTime`	Specifies the time the job (instance) was run.
`$InstanceID`	Specifies the name of the job instance.
`$SummaryItemList`	Specifies the list of items in the summary notification. Each item corresponds to a certificate the job detects for renewal.
`$SummaryTotalfailure`	Specifies the total number of items in the summary report that failed.
`$SummaryTotalNum`	Specifies the total number of items (certificates that require to be renewed) in the summary report.
`$SummaryTotalSuccess`	Specifies how many of the total number of items in the summary report succeeded.

Table 14.11 lists the tokens for the inner list items.

Table 14.11 Tokens for items in renewal-notification job's summary report

Token	Description
`$CertType`	Specifies the type of certificate—whether SSL client (`client`), SSL server (`server`), Registration Manager's signing certificate (`ra`), Certificate Manager's CA signing certificate (`ca`), router certificate (`Cisco-router`), or other (`other`).
`$HttpHost`	Specifies the fully qualified host name of the Certificate Manager or Registration Manager to which end entities should connect to renew their certificates. (The token enables you to construct the URL which end entities use to renew their certificates; see the example in "Customizing Message Templates" on page 459.)
`$HttpPort`	Specifies the port number at which the Certificate Manager or Registration Manager is listening to certificate-renewal requests from end entities. (The token enables you to construct the URL which end entities use to renew their certificates; see the example in "Customizing Message Templates" on page 459.)
`$IssuerDN`	Specifies the distinguished name of the certificate issuer.
`$NotAfter`	Specifies the `NotAfter` attribute.
`$NotBefore`	Specifies the `NotBefore` attribute.

Table 14.11 Tokens for items in renewal-notification job's summary report (Continued)

Token	Description
`$RequestorEmail`	Specifies the requestor's email address.
`$RequestType`	Specifies the request type—whether it is a certificate enrollment, certificate renewal, certificate revocation, key archival, or key recovery request.
`$SerialNumber`	Specifies the serial number of the certificate.
`$Status`	Specifies whether the operation failed or succeeded.
`$SubjectDN`	Specifies the distinguished name of the certificate subject.

Tokens for Request In Queue Notification Messages

Table 14.12 lists the tokens that you can use for formulating the content of the `RequestInQueueJob` job's summary report.

Table 14.12 Tokens for the request-in-queue job's summary report

Token	Description
`$InstanceID`	Specifies the ID assigned to the subsystem that sent this notification. • If the notification is sent by a Certificate Manager, this will be `ca`. • If the notification is sent by a Registration Manager, this will be `ra`.
`$ExecutionTime`	Specifies the time the job (instance) was run.
`$RecipientEmail`	Specifies the email address of the recipient.
`$SenderEmail`	Specifies the email address of the sender (it is the same as the one you specify in the `Sender's E-mail Address` field in "Configuring a Subsystem to Send Request Queue Notifications" on page 454).
`$SummaryTotalNum`	Specifies the total number of items (certificate requests that are pending in the queue) in the summary report.

Tokens for Directory Update Notification Messages

This section lists the tokens that are available in summary message templates for instances of the `UnpublishExpiredJob` class or plug-in module.

Table 14.13 lists the tokens that are available for this jobs's summary report. You can customize the content and format of the items in the report by using the tokens defined in Table 14.14.

Table 14.13 Tokens for the unpublish-expired job's summary report

Token	Description
`$InstanceID`	Specifies the name of the job instance that generated this summary report.
`$ExecutionTime`	Specifies the time the job (instance) was run.
`$SummaryItemList`	Specifies the list of items in the summary notification. Each item corresponds to a certificate the job detects for removal from the publishing directory.
`$SummaryTotalfailure`	Specifies the total number of items in the summary report that failed.
`$SummaryTotalNum`	Specifies the total number of items (certificates to be removed from the directory) in the summary report.
`$SummaryTotalSuccess`	Specifies how many of the total number of items in the summary report succeeded.

Table 14.14 lists the tokens for the inner list items.

Table 14.14 Tokens for items in the unpublish-expired job's summary report

Token	Description
`$CertType`	Specifies the type of certificate—whether SSL client (`client`), SSL server (`server`), Registration Manager's signing certificate (`ra`), Certificate Manager's CA signing certificate (`ca`), router certificate (`Cisco-router`), or other (`other`).
`$IssuerDN`	Specifies the distinguished name of the certificate issuer.
`$NotAfter`	Specifies the `NotAfter` attribute.
`$NotBefore`	Specifies the `NotBefore` attribute.
`$RequestorEmail`	Specifies the requestor's email address.
`$SerialNumber`	Specifies the serial number of the certificate.
`$Status`	Specifies whether the operation failed or succeeded.
`$SubjectDN`	Specifies the distinguished name of the certificate subject.

Chapter

15

Configuring Schedulable Jobs

Netscape Certificate Management System (CMS) provides a customizable Job Scheduler component that supports various mechanisms for scheduling `cron` jobs. This chapter explains how to configure Certificate Management System to use specific job plug-in modules for accomplishing jobs. The chapter also shows how plug-in implementations and configured instances for various job items appear in the configuration file.

Before reading this chapter, you should have read the chapter "Introduction to Job Scheduling and Notifications" on page 433. In particular, you should be familiar with the various job plug-in modules that come with Certificate Management System. If you are not, see "Overview of Job Plug-in Modules" on page 434.

The chapter has the following sections:

Job Management

You can manage jobs in two ways:

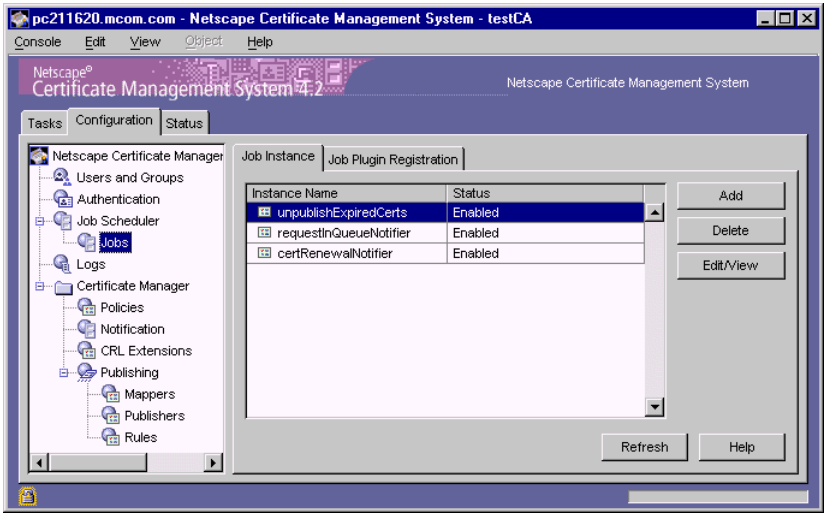
- By making changes to the job-specific parameters from the CMS window. See “Job Management From the CMS Window” on page 468.
- By editing the parameters in the configuration file. See “Job Scheduler Parameters in the Configuration File” on page 470.

The recommended method is to use the CMS window.

Job Management From the CMS Window

Figure 15.1 shows the CMS window, which provides the required user interface to support job management.

Figure 15.1 Job Scheduler information in the CMS window



In the CMS window you will find a single `Job Scheduler` object, within which is another object identified as `Jobs`. The `Jobs` object represents the job plug-in implementations and instances currently recognized by this instance of Certificate Management System; the `Job Scheduler` object represents the schedule for the configured jobs.

From the CMS window you can accomplish the following operations:

- Configuration of currently registered job plug-in modules
- Registration of new (custom) job plug-in modules

The sections that follow describe the parts of the window from which you carry out these operations.

Job Instance Tab

The Job Instance tab lists jobs that are currently scheduled for the server, so that you can manage them at a single place. From this tab you can perform the following operations:

Add. The add operation shows a list of registered job plug-in modules from which you can select the one you want to configure. When you save the changes, Certificate Management System creates the instance and displays it in the list of jobs. For instructions, see "Step 4. Add New Jobs" on page 475.

Delete. The delete operation allows you to remove unwanted jobs from the CMS configuration. For instructions, see "Step 3. Delete Unwanted Jobs" on page 475.

Edit/View. The edit operation allows you to view and modify configuration parameter values of currently configured jobs. For instructions, see "Step 2. Modify Existing Jobs" on page 472.

Job Plugin Registration Tab

The Job Plugin Registration tab lists the currently registered job plug-in modules for the selected CMS instance and gives you access to the window from which you can register new modules. On this tab you will find the names of registered modules listed on the left and the path to the Java class that implements the module listed on the right.

You can perform the following operations from this tab:

Register. This operation allows you to register a new job module. When you save the changes, Certificate Management System loads the plug-in module implementation for the job and displays it in the list of currently registered plug-in modules. For instructions, see "Registering a Job Module" on page 482.

Delete. This operation allows you to remove unwanted job modules from the CMS framework. For instructions, see "Deleting a Job Module" on page 484.

Job Scheduler Parameters in the Configuration File

The sample configuration file shown on page 101 shows how Job Scheduler-specific information appears in the configuration file. Keep the following points in mind:

- All job-specific information, such as registered job modules and configured instances, appears in the Job Scheduler section of the configuration file.
- Each registered job module is identified by its implementation name and the corresponding Java class.
- Each job (or configured instance of a job module) is identified by the name specified when the job was created.
- You can create as many instances of an implementation as you like; each instance must have a unique name.

To change the configuration by editing the configuration file, follow the instructions in "Changing the Configuration by Editing the Configuration File" on page 86.

Scheduling Automated Jobs

You can configure the Certificate Manager and Registration Manager to run automated jobs, that is execute specific jobs at specified times. This section explains how to configure a subsystem to evaluate end-entity requests based on a set of policy rules. The steps are as follows:

This section explains how to use the CMS window to perform the following operations:

- Step 1. Plan
- Step 2. Modify Existing Jobs
- Step 3. Delete Unwanted Jobs
- Step 4. Add New Jobs
- Step 5. Schedule the Frequency
- Step 6. Customize Message Templates
- Step 7. Verify Mail Server Settings

For information on adding or changing job-specific information in the configuration file, see "Job Scheduler Parameters in the Configuration File" on page 470.

Step 1. Plan

Before configuring a Certificate Manager or Registration Manager to run jobs, be sure to do the following:

- Read "Overview of Job Plug-in Modules" on page 434, and determine the jobs you want the server to run. Jobs that you might want to schedule include email notifications of timed events (such as the expiration of a certificate) that require action on the part of users, and periodic activities such as removing expired certificates from the publishing directory.
- Read "Customizing Notification Messages" on page 455 to get familiar with the templates the server uses for formulating notification messages. If you want to customize them, determine the tokens you want to use.

Step 2. Modify Existing Jobs

Modifying a job involves changing the configuration parameter values of the job instance; you cannot change the name of a job. To change the name of a job, create a new job using the same job plug-in module (that you used to create the job you want to rename) with the same parameter values, and delete the old one.

As a part of modifying a job, you can change its status from enabled to disabled or vice versa by checking or unchecking the `enable` parameter. A subsystem executes only those jobs that are enabled.

During installation, the Certificate Manager and Registration Manager automatically create a set of jobs (that you would most likely want to use) using the job plug-in modules registered by default. Figure 15.2 shows the jobs created for a Certificate Manager. The Registration Manager also has a similar list. Table 15.1 summarizes the default jobs created for both Certificate Manager and Registration Manager.

After installation, you must verify whether you want to use these jobs, check how these jobs are configured, and make the appropriate configuration changes. If you don't want to use a job, delete it from the configuration following the instructions in "Step 3. Delete Unwanted Jobs" on page 475; alternatively, you may keep it in the disabled state. If you want to create a new job, follow the instructions in "Step 4. Add New Jobs" on page 475.

Figure 15.2 Default jobs created for a Certificate Manager

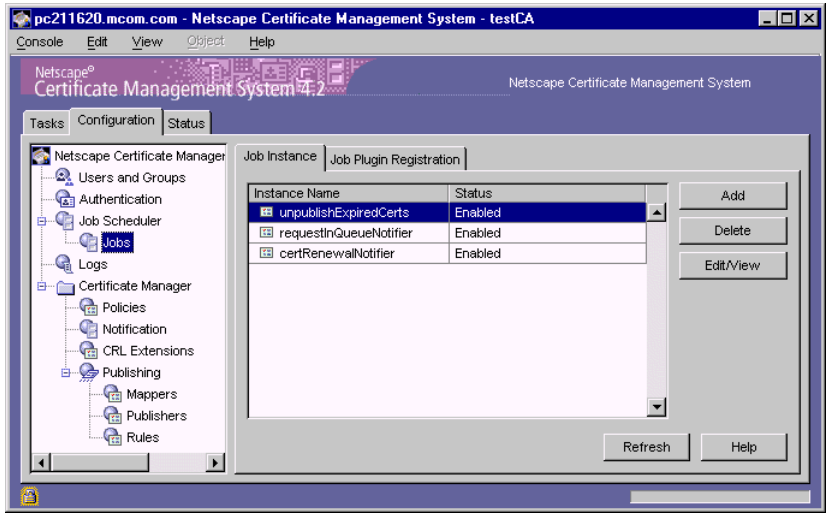


Table 15.1 Default jobs created for a Certificate Manager and Registration Manager

Job name	Created for Certificate Manager	Created for Registration Manager
`certRenewalNotifier` See "Certificate Renewal Notifications" on page 435.	Yes	Yes
`requestInQueueNotifier` See "Directory Update and Notification" on page 444.	Yes	Yes
`unpublishExpiredCerts` See "Directory Update and Notification" on page 444.	Yes	No

To modify a configured job in the CMS configuration:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Job Scheduler, then select Jobs.

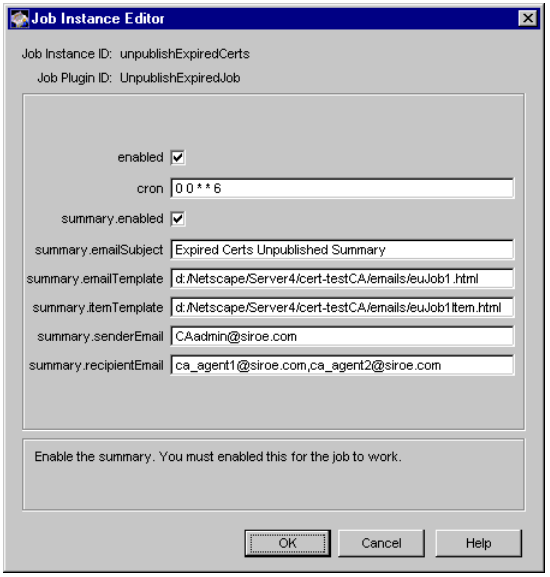
The Job Instance tab appears (Figure 15.2). The default jobs are listed in Table 15.1.

4. In the Instance Name list, select a job that you want to modify.

   For the purposes of this instruction, assume that you selected the job named `unpublishExpiredCerts`.

5. Click Edit/View.

   The Job Instance Editor window appears, showing how this job is currently configured. An example is shown below.



6. Make the necessary changes and click OK.

7. Repeat steps 4 through 6 for the remaining jobs.

8. Click Refresh.

Step 3. Delete Unwanted Jobs

You can delete unwanted jobs from the CMS configuration, by using the CMS window. If you think you might need a job in the future, instead of deleting it from the configuration you should disable it by setting the `enable` parameter

value to `false`. In this way, you can avoid re-creating the job in the future. Because Certificate Management System executes only those jobs that are currently enabled, keeping unwanted jobs in a disabled state in the configuration does not affect the server's functioning.

To delete a job from the CMS configuration:

1. In the Job Instance tab, select the job you want to delete and click Delete.
2. When prompted, confirm the delete action.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. Don't restart the server yet; you can do so after you've made all the required changes.

Step 4. Add New Jobs

Adding a job to the CMS configuration involves creating a new instance of an already registered plug-in module, assigning a unique name (an alphanumeric string with no spaces) for the instance, and entering appropriate values for the parameters that define the plug-in module you want to create an instance of. When you add a job, the CMS configuration is updated with the appropriate information.

When naming a job, be sure to formulate the name using any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-); other characters and spaces are not allowed. For example, you can type `My_Job` or `MyJob` as the instance name, but not `My Job`.

Figure 15.3 shows the job modules registered with a Certificate Manager. The Registration Manager also has a similar list. Table 15.2 summarizes the default modules registered with both Certificate Manager and Registration Manager. If you have registered any custom job modules (see "Registering a Job Module" on page 482), they too will be available for selection.

Figure 15.3 Default job modules registered with a Certificate Manager

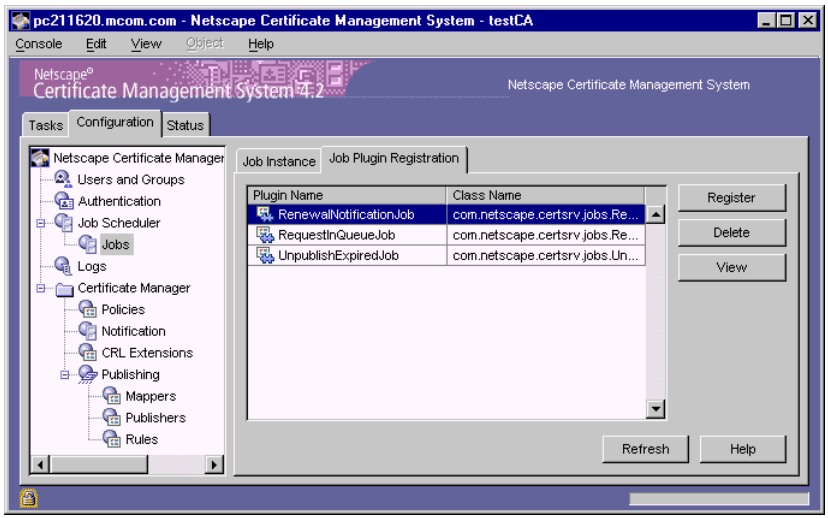


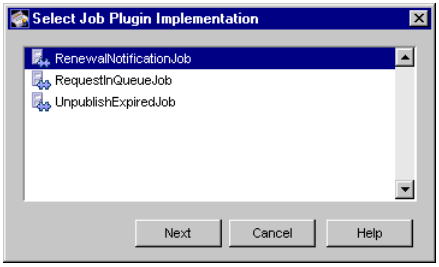
Table 15.2 Job modules registered with a Certificate Manager and Registration Manager

Job plug-in module name and description	Provided with Certificate Manager	Provided with Registration Manager
`RenewalNotificationJob` For details, see "Certificate Renewal Notifications" on page 435.	Yes	Yes
`RequestInQueueJob` For details, see "Notification of Request Queue Status" on page 440.	Yes	Yes
`UnpublishExpiredJob` For details, see "Directory Update and Notification" on page 444.	Yes	No

To add a job to the CMS configuration:

1. In the Job Instance tab, click Add.

   The Select Job Plugin Implementation window appears. It lists the currently registered job modules. The default modules are listed in Table 15.2.

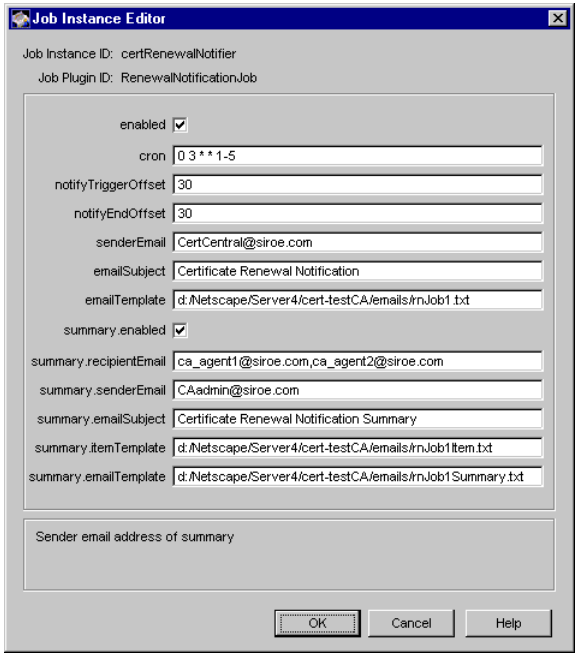


2. Select a module.

   For the purposes of this instruction, assume that you selected the `RenewalNotificationJob` module.

3. Click Next.

The Configure Job Instance Parameters window appears. It lists the configuration information required for this job.



4. Enter the appropriate information.

   Job Instance ID. Type a unique name that will help you identify the job. Be sure to formulate the name using any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-). For example, you can type `My_Job` or `MyJob` as the instance name, but not `My Job`.

   enabled. To enable the job, type `true`; to disable the job, type `false`.

   cron. Specifies the cron specification for when this job should be run. In other words, it specifies the time at which the Job Scheduler daemon thread should check the certificates for sending renewal notifications. For example, `03**1-5`. Be sure to follow the convention specified in “Schedule for Executing Jobs” on page 448

   notifyTriggerOffset. Type the number of days before certificate expiration the first notification should be sent. For example, if you want the server to send renewal notifications to users 30 days before their certificates expire, type 30.

notifyEndOffset. Type the number of days after the certificate expire notifications will continue to be sent, if the certificate is not renewed. For example, if you want the server to continue sending renewal notifications to users (if they don't renew their certificates) 30 days after their certificates expire, type 30.

senderEmail. Type the complete email address to which the server should send notifications regarding any delivery problems. For example, `CertCentral@siroe.com`.

emailSubject. Type the subject line of the notification message; the subject line must be an alphanumeric string of up to 255 characters. For example, `Certificate Renewal Notification`.

emailTemplate. Type the path, including the filename, to the directory that contains the template to be used for formulating the message content. For example: `C:/Netscape/Server4/cert-testCA/emails/renewJob.txt`.

summary.enabled. Type `true` if you want the server to compile a summary report of renewal notifications and send. Type `false` if you don't want the server to compile a summary report of renewal notifications.

summary.recipientEmail. Type the email addresses of recipients of the summary report; when specifying multiple recipients, separate addresses by commas. These can be, for example, agents who need to know the status of user certificates. For example, `ca_agent1@siroe.com, ca_agent2@siroe.com`.

summary.senderEmail. Type the full email address of the sender (of the summary message); in case of a delivery problem, the server will send a notification to this address. For example, `CAadmin@siroe.com`.

summary.emailSubject. Type the subject line of the summary message; the subject line must be an alphanumeric string of up to 255 characters. For example, `Certificate Renewal Notification Summary`.

summary.itemTemplate. Type the path, including the filename, to the directory that contains the template to be used for formulating the content and format of each item to be collected for the summary report (see the `summary.emailTemplate` parameter below). For example, `C:/Netscape/Server4/cert-testCA/emails/renewJobItem.txt`. For more information, see For details, see "Customizing Notification Messages" on page 455.

summary.emailTemplate. Type the path, including the filename, to the directory that contains the template to be used for formulating the summary report. For example, `C:/Netscape/Server4/cert-testCA/emails/renewJobSummary.txt`. For more information, see "Customizing Notification Messages" on page 455.

5. Click OK.

   You are returned to the Policy Rules Management tab.

6. Repeat steps 1 through 5 and create additional rules, if required.

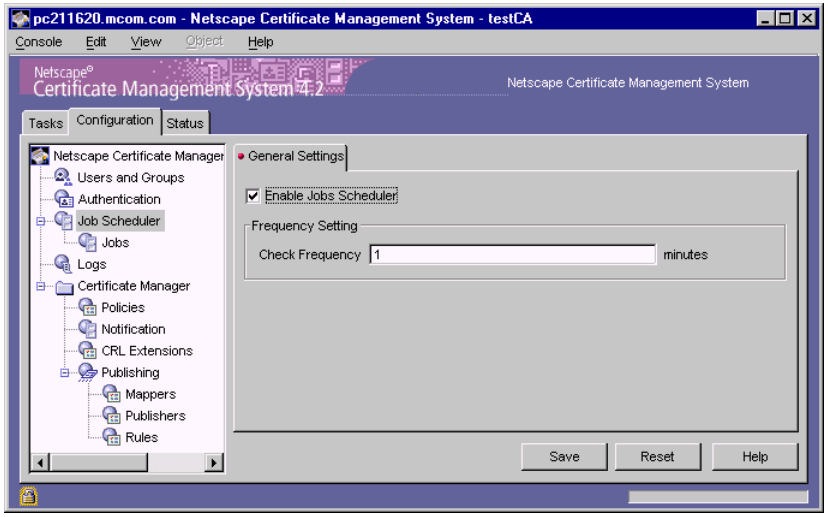
Step 5. Schedule the Frequency

The Certificate Manager and Registration Manager can execute a job only if the Job Scheduler is turned on (or enabled). As a part of turning the Job Scheduler on, you also specify the frequency at which the Job Scheduler daemon should check if any of the configured jobs need to be executed.

To schedule the interval for executing the job:

1. In the navigation tree, click Job Scheduler.

   The General Settings tab appears. It shows whether the Job Scheduler component is currently enabled or disabled.



2. Enter information as appropriate:

   Enable Job Scheduler. Check this option to enable the Job Scheduler. To disable the Job Scheduler uncheck the option; disabling turns off all the jobs.

   Check Frequency. Type the frequency at which the Job Scheduler daemon thread should wake up and call the configured jobs that meet the cron specification (see "Schedule for Executing Jobs" on page 448). By default, it is set to one minute.

3. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Step 6. Customize Message Templates

Each job uses templates for formulating the notification-message and summary-message contents. Make sure to customize the appropriate templates to suit your organization's requirements. For details on default templates, see "Customizing Notification Messages" on page 455.

Step 7. Verify Mail Server Settings

The Certificate Manager and Registration Manager use the mail server specified in the SMTP tab (of the CMS window) for routing or sending email notifications automatically. Verify that the host name and port number of the mail server are accurate; see "SMTP Settings" on page 168.

Managing Job Plug-in Modules

This section explains how to use the CMS window to perform the following operations:

- Registering a Job Module
- Deleting a Job Module

For information on adding or changing job-specific information in the configuration file, see "Job Scheduler Parameters in the Configuration File" on page 470.

Registering a Job Module

You can register custom job plug-in modules from the CMS window. Registering a new module involves specifying the name of the module and the full name of the Java class that implements the module. For example, you can add a job implementation named as follows:

```
com.netscape.jobscheduler.unpublishUserCert
```

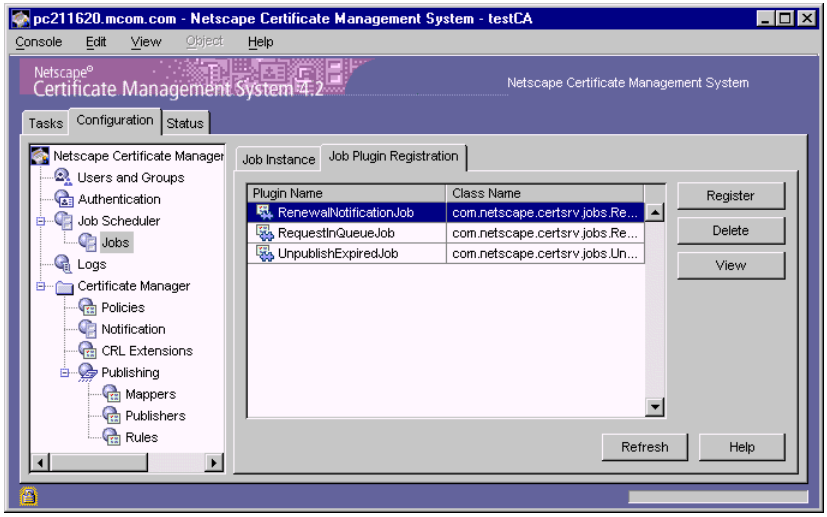
Before registering a module, be sure to put the Java class for the module in the `classes` directory (the implementation must be on the class path).

To register a job module in the CMS framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Job Scheduler, then select Jobs.

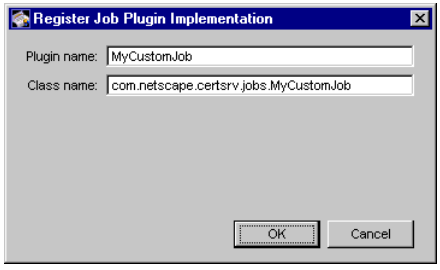
   The Job Instance tab appears. It lists any currently configured jobs.
4. Select the Job Plugin Registration tab.

The Job Plugin Registration tab appears. It lists currently registered job modules.



5. Click Register.

   The Register Job Scheduler Plugin Implementation window appears.



6. Specify information as appropriate:

   Plugin name. Type a name for the plug-in module.

   Class name. Type the full name of the class for this module—that is, the path to the implementing Java class. If this class is part of a package, be sure to include the package name. For example, if you are registering a class named `myJob` and if this class is in a package named `com.myCompany`, type `com.myCompany.myJob`.

7. Click OK.

The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Deleting a Job Module

You can delete unwanted job plug-in modules by using the CMS window. Before deleting a module, be sure to delete all the instances that are based on this module; for instructions, see "Step 3. Delete Unwanted Jobs" on page 475.

To delete a job module from the CMS framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Job Scheduler, then select Jobs.

   The Job Instance tab appears. It lists any currently configured instances.
4. Select the Job Plugin Registration tab.

   The Job Plugin Registration tab appears. It lists currently registered job modules.
5. In the Plugin Name list, select the module you want to delete and click Delete.
6. When prompted, confirm the delete action.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

6

Policies

Chapter

16

Introduction to Policy

You can configure the main subsystems of Netscape Certificate Management System (CMS)—the Certificate Manager, Registration Manager, and Data Recovery Manager—to apply certain organizational policies on an end entity's certificate enrollment and management requests before servicing them. For example, some of the policies you might want a Certificate Manager to impose on these requests may include setting a minimum and maximum limit on validity period and key length of certificates, setting extensions based on the end entity's role within an organization, setting signing algorithms, and so on.

This chapter provides an overview of policy in general, and it explains the various policy plug-ins supported out of the box.

The chapter has the following sections:

What Is Policy?

Policy refers to a set of rules that Certificate Management System uses to evaluate or verify an incoming request from an end entity and to determine the outcome; the incoming requests that are governed by policies include certificate issuance, certificate renewal, certificate revocation, key archival, and key recovery requests. For example, in the case of a certificate issuance request, the outcome would be the certificate content.

- A Certificate Manager's policy can include rules for evaluating certificate formulation, signing, renewal, and revocation requests. For example, you can configure a Certificate Manager's policy to impose restrictions on validity length, key type, key length, subject name, extensions, and signing algorithm during certificate issuance.

- A Registration Manager's policy can include rules for verifying incoming certificate issuance, renewal, and revocation requests from end entities in order to formulate the certificate content before forwarding the requests to a Certificate Manager for signing. For example, you can configure a Registration Manager's policy to impose restrictions on validity period, key length, subject name, and extensions. In general, policies for Registration Manager are largely the same as for Certificate Manager.

- A Data Recovery Manager's policy can include rules for verifying users' encryption private key archival and recovery requests.

Using policies, you can configure Certificate Management System to perform one or more of the following operations on each certificate issuance or management request it receives:

- Screen the request for specific content, and modify, reject, or defer (for agent approval) it accordingly. For example, the request might be checked for the inclusion of organizational constraints, such as key algorithm, key size, validity period, or a particular signing algorithm; if it did not meet the requirement, the subsystem would modify the request or return an error, depending on the severity of the problem.

- Set common attributes, such as extensions for user and server certificate requests.

Policy Rules

A policy rule refers to a uniquely configured instance of any policy plug-in implementation; see “Built-in Policy Plug-in Modules” on page 499 for a list of plug-in modules provided for Certificate Management System. For example, you can use the plug-in module provided for setting validity periods on certificates to configure a policy rule that forces validity periods for all client certificates issued by a Certificate Manager to fall within a predetermined range, say between 6 and 24 months. A subsystem’s policy configuration can consist of one or more policy rules, each performing one or more of the following operations:

- Validate the request content by comparing it with configured criteria; reject, modify, or defer (for agent approval) the request if any of the request parameters are invalid.
- Build certificate content—for example, set common extensions and the validity period.
- Enforce organizational constraints, such as subject name, key algorithm, key size, and validity period.
- Determine whether the private key should be archived.

Keep in mind that the server applies the rules when processing end-entity requests and after agent approval (for deferred requests).

Types of Policy Rules

Certificate Management System supports distinct policy rules for each of the operations that end entities perform—certificate enrollment, renewal, and revocation, and key archival and recovery. Consequently, there are five broad categories of policies, corresponding to these types of operations:

- Enrollment policies
- Renewal policies
- Revocation policies
- Key archival policies

- Key recovery policies

To facilitate this classification, Certificate Management System supports a parent interface for a generic policy rule and other operation-specific interfaces that extend the parent interface. Check the CMS SDK, available in the form of Java Docs at this location: `<server_root>/cms_sdk/sdkdocs`

For general guidelines on developing custom policy modules and adding them to the CMS policy framework, take a look at the samples installed at this location: `<server_root>/cms_sdk/samples/policy`

Using Predicates in Policy Rules

You can use predicates in a policy rule. A predicate indicates whether the rule that contains the predicate applies to a request. If you specify a predicate as part of the rule configuration, the policy rule applies that predicate based on request attributes to determine whether the rule is applicable for a request.

The policy predicate is a logical expression. You form the expression using variables and relational operators (`AND` or `OR`). For example, you could set up a predicate to put the CRL Distribution Point extension only in SSL client certificates, or set different validity dates for certificates for users in different groups.

The following are sample predicates:

```
HTTP_PARAMS.certType==client AND HTTP_PARAMS.ou==Engineering
```

```
HTTP_PARAMS.certType==server AND HTTP_PARAMS.o==Siroe OR
HTTP_PARAMS.certType==ca
```

Expression Support for Predicates

You form an expression using an attribute, its value, and one or more of the operators listed in Table 16.1. For a list of attributes, see “Attributes for Predicates” on page 493.

Table 16.1 Predicates in policy: supported comparison and logical operators

Operator	Description
`==`	Equal to
`!=`	Not equal to
`AND`	Logical operator `AND`
`OR`	Logical operator `OR`

Note that the expression parsing support currently supports only two comparison operators (`==`, `!=`) and two relational operators (`AND`, `OR`).

Policy expressions are formed with the following rules:

```
PrimitiveExpression | AndExpression | OrExpression
```

- `PrimitiveExpression` is equal to: Attribute `op` Value

  where

  Attribute can be a string

  `op` can be any of these operators: `==` or `!=`

  Value can be a string

- `AndExpression` is equal to: Expression `AND` Expression
- `OrExpression` is equal to: Expression `OR` Expression

In an expression, the `AND` operator takes precedence over an `OR` operator. For example, the expression

```
HTTP_PARAMS.certType==client AND HTTP_PARAMS.ou==Engineering OR
HTTP_PARAMS.certType==ca
```

is interpreted as

```
(HTTP_PARAMS.certType==client AND HTTP_PARAMS.ou==Engineering)
OR HTTP_PARAMS.certType==ca
```

Certificate Management System evaluates an expression based on the attributes in the request. The attributes are filled in by servlets from the HTTP input forms used for request submission. Some attributes, such as passwords typed in the form are not stored in the request. Other attributes regarding the end entity, such as the user ID, are set on the request after successful authentication. The servlets also interpret the form content, for example, retrieving the key material out of the `KEYGEN` or `PKCS #10` information and setting the key in the certificate content. They can also set additional attributes related to the certificate content on the request. In general, you can configure which attributes—for example, sensitive attributes such as passwords—should or shouldn't be stored in the request.

Note that all data related to an end entity is gathered at the servlet level and set on the request before the request is passed to the policy subsystem. The policy subsystem applies configured policy rules on the request, determines whether the request needs agent approval, performs constraint- and extension-specific checks on the request attributes, and then formulates the certificate content by adding the appropriate information, such as the validity period and extensions.

The expression queries the request for the attributes, compares the value returned with the value provided in the predicate, and returns a boolean result.

Be aware that if the same name is in a HTTP form input and authentication token (authentication result) the authentication result can override the HTTP form input. For example, if `email` is in a HTTP input and an authentication module also puts `email` in the authentication result (that is, authtoken) the `email` value from the authentication module will override the `email` value from the HTTP input in the request. A predicate using `email` in an expression will be evaluated to the value of the authentication instead of the HTTP input value.

The following are sample predicates:

```
HTTP_PARAMS.certType==client AND HTTP_PARAMS.ou==Engineering
```

```
HTTP_PARAMS.certType==server AND HTTP_PARAMS.o==Siroe OR
HTTP_PARAMS.certType==ca
```

Attributes for Predicates

Attributes for predicates can come from any of the following:

- Input form—that is, the HTML form that end entities use for submitting certificate requests.
- Authentication token—what the authentication subsystem returns after successfully authenticating an end entity.
- A service—for example, a Certificate Manager, Registration Manager, or Data Recovery Manager service can add certain attributes to the end-entity request.
- Policy processor—what the policy subsystem returns after subjecting the end-entity request to policy checking. For example, an extension-based policy can set an appropriate extension in the certificate.

Table 16.2 lists default attributes that are supported by various request object implementations.

Table 16.2 Attributes supported by request object implementations

Request type	Variable name	Description
Default attributes from an input form:		
Enrollment	`certType`	Specifies the certificate type. Default values include the following: • `ca` (Certificate Manager's CA signing certificate) • `CEP-Request` (router certificate) • `client` (client certificates) • `objSignClient` (Object Signing certificates • `ocspResponder` (OCSP Responder certificate) • `other` • `ra` (Registration Manager's signing certificate) • `server` (server certificate)

Table 16.2 Attributes supported by request object implementations (Continued)

Request type	Variable name	Description
Enrollment	`requestFormat`	Specifies the certificate request format. Default values include the following: • `keygen` • `pkcs10` • `clientAuth`
Enrollment	`doSslAuth`	Specifies whether the client is required to do SSL client authentication during enrollment. Default values include the following: • `on` • `off`
Enrollment	`certauthEnroll`	Specifies whether it is a certificate-based enrollment (see "Certificate-Based Enrollment" on page 357). Default values include the following: • `on` • `off`
Enrollment	`certauthEnrollType`	Specifies the number of keys to be generated for a certificate-authenticated enrollment—whether a single signing key, a single encryption key, or dual keys (one for signing and another for encryption) is to be generated. Default values include the following: • `single` • `encryption` • `dual`
Enrollment	`cepsubstore`	Specifies the name of the CEP service; for example, `cep1` and `cep2`. When setting up multiple CEP services, you can use predicates to differentiate one service for another; see "Step 4. Set Up Multiple CEP Services" on page 1103.
Enrollment, Renewal, and Revocation	`requestStatus`	Specifies when (or the phase in which) a request gets subjected to policy processing: • `begin` specifies that the request be subjected to a policy before it gets queued for agent approval. • `pending` specifies that the request be subjected to a policy after agent approval.

Table 16.2 Attributes supported by request object implementations (Continued)

Request type	Variable name	Description
Renewal	`requestFormat`	Specifies the certificate request format. Default values include the following: • `clientAuth` • `pkcs10`
Default attributes from an authentication token: (Upon successful authentication these attributes go into an enrollment request)		
Enrollment	`authMgrImplName`	Specifies the name of the authentication plug-in module that authenticated the request.
Enrollment	`authMgrInstName`	Specifies the name of the authentication instance that authenticated the request.

You can define your own attributes for predicates, if there's a need. For example, assume you have two organizational units Sales and Manufacturing and you want to issue client certificates with different validity periods to users in these two units. A quick and easy way to accomplish this would be to define a new attribute for the organizational unit, add the attribute to the enrollment form that the users in these organizational units use for certificate enrollment (so that the server receives it from the HTTP input), and use the attribute in the predicate expression for the validity constraints policy—a policy rule that determines the validity period of certificates the server issues. For details on this policy, see "Validity Constraints Policy" on page 543.

Note that to define a new attribute in any of the HTML forms, all you need to do is to add the following line to the corresponding HTML form:

```
<input type="HIDDEN" name="attribute_name"
value="attribute_value">
```

Assuming that the new attribute you define for the organizational unit is `orgunit`, the line you would add to the enrollment form would be:

```
<input type="HIDDEN" name="orgunit" value="Sales">
```

To add this line to an enrollment form, you would:

1. Open the corresponding HTML file in a text editor.
2. Locate the section that lists the HTTP input variables.
3. Add this line: `<input type="HIDDEN" name="orgunit" value="Sales">`
4. Save your changes and close the file.

For the server to use the attribute (to distinguish enrollment requests from users in the Sales unit versus those in the Manufacturing unit) to issue certificates with the appropriate validity periods, you must formulate your predicate expression with the attribute you added. Here's how you do this:

1. Create a new instance of the `ValidityConstraints` policy plug-in implementation.
2. Enter the appropriate values for all the attributes. Assume you
   - named the instance `ValidityRule1`
   - set the minimum validity period to 10 days
   - set the maximum validity period to 180 days
   - defined the predicate expression as `HTTP_PARAMS.certType==client AND HTTP_PARAMS.orgunit==Sales`

     (This expression specifies that the policy be applied to only client certificate requests from users in the organizational unit named Sales.)

   A sample of the resulting configuration entries in the CMS configuration file would be as follows:

```
ca.Policy.rule.ValidityRule1.enable=true

ca.Policy.rule.ValidityRule1.implName=ValidityConstraints

ca.Policy.rule.ValidityRule1.maxValidity=180

ca.Policy.rule.ValidityRule1.minValidity=10

ca.Policy.rule.ValidityRule1.predicate=HTTP_PARAMS.certType=
=client AND HTTP_PARAMS.orgunit==Sales
```

Now, for setting the validity period in certificates of users who are not in the Sales organization—in this case, this would be Manufacturing—you would create another instance of `ValidityConstraints` policy rule as before with a different set values.

Assume you

- named the instance `ValidityRule1`
- set the maximum validity period to 60 days
- set the minimum validity period to 10 days
- defined the predicate expression as `HTTP_PARAMS.certType==client AND HTTP_PARAMS.orgunit!=Sales`

  (This expression specifies that the policy be applied to only client certificate requests from users who are not in the organizational unit named Sales.)

A sample of the resulting configuration entries in the CMS configuration file would be as follows:

```
ca.Policy.rule.ValidityRule2.enable=true

ca.Policy.rule.ValidityRule2.implName=ValidityConstraints

ca.Policy.rule.ValidityRule2.maxValidity=60

ca.Policy.rule.ValidityRule2.minValidity=10

ca.Policy.rule.ValidityRule2.predicate=HTTP_PARAMS.certType=
=client AND HTTP_PARAMS.orgunit!=Sales
```

The new configuration would result in certificates with a validity period of six months for users in the Sales organizational unit and a validity period of three months for users in the Manufacturing unit.

Policy Processor

Each subsystem—the Certificate Manager, Registration Manager, or Data Recovery Manager—has its own policy processor. Each processor subjects an incoming request to the applicable policy rules for that subsystem.

When a subsystem starts up, its policy processor reads the current policy configurations from the configuration file, initializes them, and classifies them based on their type (see "Types of Policy Rules" on page 489). Then, when the subsystem receives an authenticated request, its request processor invokes the policy processor to apply policies on that request. The policy processor applies the rules on the request based on the request type. The policy processor also filters the rules based on predicates (see "Using Predicates in Policy Rules" on page 490).

Note that the policy processor applies only the *enabled* policy rules, in the order in which they are configured, before determining the final outcome. Each rule the processor executes returns a `PolicyResult` object. Three return values are possible:

- `PolicyResult.REJECTED` (indicates that the request failed the rule)
- `PolicyResult.DEFERRED` (indicates that the request requires agent approval)
- `PolicyResult.ACCEPTED` (indicates that the request passed the rule)

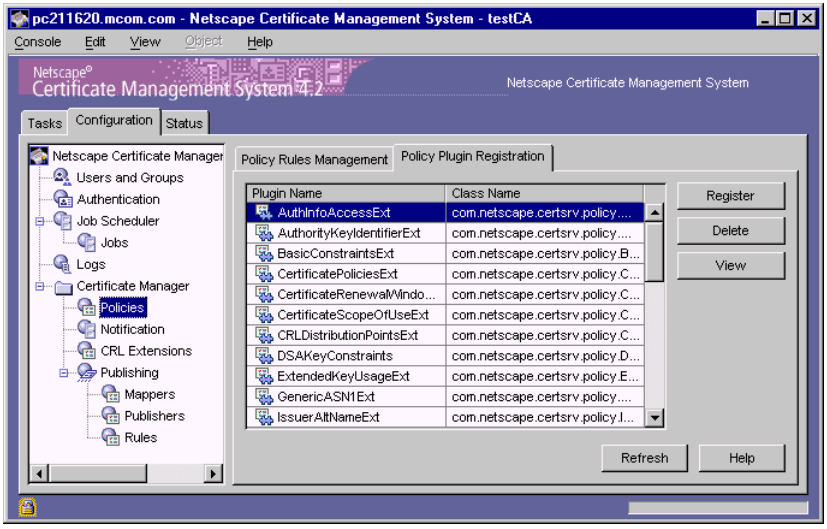
After all the policy rules are applied, the processor determines the status of the request (in this order):

1. If the request failed any policy rule (that is, if any of the policy rules returned a `PolicyResult.REJECTED` value), the processor rejects the request. The rule that rejected the request sets appropriate error messages on the request.
2. If at least one of the policy rules requires agent approval for the request (that is, if any of the policy rules returned a `PolicyResult.DEFERRED` value), the processor stores the request in the request queue for agent approval.
3. If the request passes all the policy rules (that is, all policy rules returned a `PolicyResult.ACCEPTED` value), the request gets serviced—for example the certificate is issued or renewed.

Built-in Policy Plug-in Modules

Certificate Management System comes with various policy plug-in modules that help you govern its certificate generation and management operations. The CMS window lists all the plugins and the corresponding classes that are currently recognized by the server; see Figure 16.1

Figure 16.1 Built-in policy plug-in modules



The modules provided out of the box can be categorized, based on their functionality, into two groups:

- Constraints-specific policy plug-in modules
- Extension-specific policy plug-in modules

The chapters that follow cover both the groups in detail. Note that Certificate Management System doesn't come with key archival and recovery policy modules.

Chapter

17

Constraints-Specific Policy Modules

You can configure Certificate Management System to apply certain organizational policies to an end entity's certificate enrollment, renewal, and revocation requests before servicing them. For example, some of the policies you might want Certificate Management System to apply to these requests may include setting a minimum and maximum limit on validity period and key length of certificates, setting extensions based on the end entity's role within an organization, setting signing algorithms, and so on.

Certificate Management System comes with various policy plug-in modules that define the formulation of a certificate's content and govern the server's certificate generation and management operations. The modules are categorized, based on their functionality, into two groups: constraints-specific policy modules and extension-specific policy modules.

This chapter explains the constraints-specific policy plug-in modules in detail—it lists and briefly describes the modules that are installed with Certificate Management System, and then explains each one in detail. For details on extension-specific modules, see "Extension-Specific Policy Modules" on page 549.

The chapter has the following sections:

Overview of Constraints-Specific Policy Modules

Constraints-specific policy plug-in modules help you define rules or constraints that Certificate Management System uses to evaluate an incoming certificate enrollment, renewal, or revocation request. Each module enables you to configure the server to check the request for particular attributes, and, based on the configured criteria, either modify these attributes or reject the request altogether.

Policy plug-in modules are implemented as Java classes and are registered in the CMS policy framework. The Policy Plugin Registration tab of the CMS window (Figure 17.1) lists all the modules that are registered with a CMS instance.

Figure 17.1 CMS window showing policy modules registered with a Certificate Manager

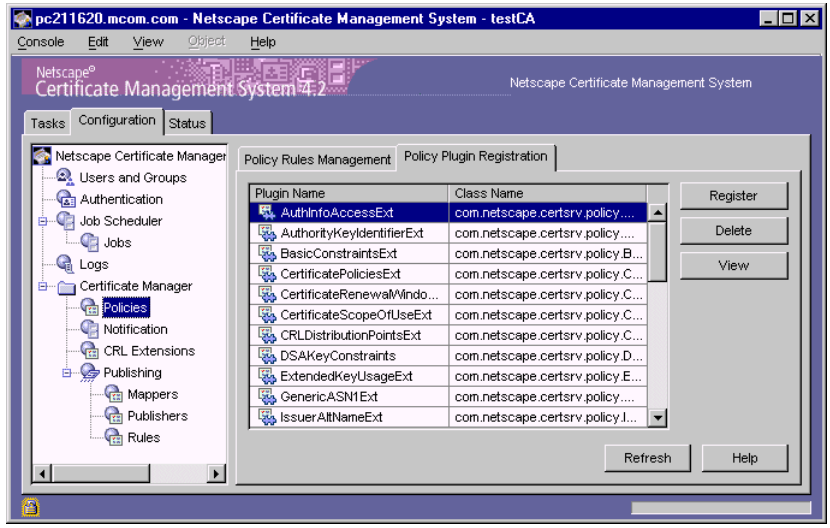


Table 17.1 lists constraints-specific policy modules that are installed with a Certificate Manager. An installation of a Registration Manager also includes all these modules, expect for the ones noted below:

- `IssuerConstraints`
- `SubCANameConstraints`
- `UniqueSubjectNameConstraints`

You can use whichever modules you need in order to define policy rules for a Certificate Manager or Registration Manager. Note that no modules are provided for the Data Recovery Manager. Both Certificate Manager and Registration Manager subject a request to policy checking as explained in “Policy Processor” on page 497. Keep in mind that the changes made to a request by a Registration Manager may be overwritten by a Certificate Manager when it subjects the request to its own policy checks.

Table 17.1 Default constraints-specific policy plug-in modules

Plug-in module name	Function
`DSAKeyConstraints`	Certifies only those DSA keys that have specific key lengths. For details, see “DSA Key Constraints Policy” on page 505.

Table 17.1 Default constraints-specific policy plug-in modules (Continued)

Plug-in module name	Function
`IssuerConstraints`	Checks for certificates that have been issued by a particular CA. For details, see "Issuer Constraints Policy" on page 509.
`KeyAlgorithmConstraints`	Certifies only those keys that are generated using one of the permitted algorithms, such as RSA or DSA. For details, see "Key Algorithm Constraints Policy" on page 512.
`PinPresentConstraints`	Checks for the presence of end users' PINs in the authentication directory when they request a certificate and accordingly approves or rejects the request. For details, see "PIN Present Policy" on page 514.
`RenewalConstraints`	Allows or rejects requests for renewal of expired certificates. For details, see "Renewal Constraints Policy" on page 518.
`RevocationConstraints`	Allows or rejects requests for revocation of expired certificates. For details, see "Revocation Constraints Policy" on page 522.
`RenewalValidityConstraints`	Enforces the number of days before which a currently active certificate can be renewed and sets a new validity period for the renewed certificate. For details, see "Renewal Validity Constraints Policy" on page 525.
`RSAKeyConstraints`	Certifies only those RSA keys that have specific key lengths. For details, see "RSA Key Constraints Policy" on page 529.
`SigningAlgorithmConstraints`	Specifies the signature algorithm to be used by the CA (a Certificate Manager) to sign certificates. For details, see "Signing Algorithm Constraints Policy" on page 533.
`SubCANameConstraints`	Checks for issuer name uniqueness and prevents a CA from issuing a subordinate CA certificate with issuer name same as its own. For details, see "Subordinate CA Name Constraints Policy" on page 536.
`UniqueSubjectNameConstraints`	Checks for certificate subject name uniqueness and prevents issuance of multiple certificates with same subject names. For details, see "Unique Subject Name Constraints Policy" on page 539.
`ValidityConstraints`	Checks whether the validity period of a certificate falls within a specific validity period. For details, see "Validity Constraints Policy" on page 543.

DSA Key Constraints Policy

The `DSAKeyConstraints` plug-in module implements the DSA key constraints policy. This policy imposes constraints on the following:

- The minimum and maximum sizes for keys
- The sizes of exponents

The policy restricts the key size to one of the sizes, such as 512 or 1024, supported by Certificate Management System.

You may apply this policy to end-entity certificate enrollment and renewal requests. For example, if you want your CA to certify public keys up to 512 bits in length for end users and 1024 for servers, you can configure Certificate Management System to do so using the policy.

During installation, Certificate Management System automatically creates an instance of the DSA key constraints policy. See "DSAKeyRule Rule" on page 508.

DSAKeyConstraints Module

The Java class that implements the DSA key constraints policy is as follows:

```
com.netscape.certsrv.policy.DSAKeyConstraints
```

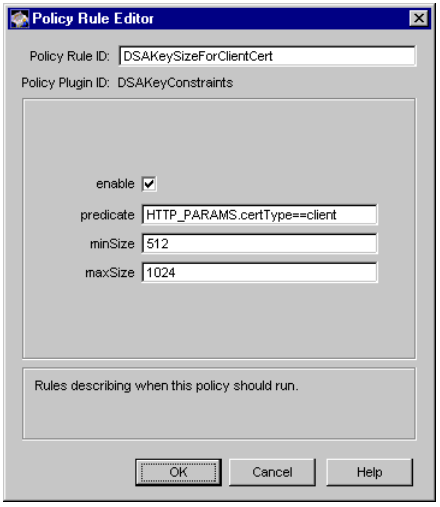
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.DSAKeyConstraints.class=com.netscape.certsrv.policy.DSAKeyConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `DSAKeyConstraints`

Figure 17.2 shows how configurable parameters for the `DSAKeyConstraints` module are displayed in the CMS window.

Figure 17.2 Parameters of the DSAKeyConstraints module



The configuration shown in Figure 17.2 creates a policy rule named `DSAKeySizeForClientCert`, which enforces a rule that the server should restrict the minimum and maximum key sizes for all DSA key-based client certificates to 512 and 1024, respectively.

Table 17.2 describes each of the parameters.

Table 17.2 Description of parameters defined in the DSAKeyConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server applies the rule to certificates specified by the predicate expression. • If you disable the rule, the server does not apply the rule to certificates.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see “Using Predicates in Policy Rules” on page 490. Example: `HTTP_PARAMS.certType==client`

Table 17.2 Description of parameters defined in the DSAKeyConstraints module (Continued)

Parameter	Description
`minSize`	Specifies the minimum length, in bits, for the key (the length of the modulus in bits). The value must be smaller than or equal to the one specified by the `maxSize` parameter. In general, a longer key size results in a key pair that is more difficult to crack. You may want to enforce a minimum length to ensure a minimum level of security. Permissible values: `512` or `1024`. You may also enter a custom key size that is between 512 and 1024, in increments of 64 bits. The default value is 512. Example: `512`
`maxSize`	Specifies the maximum length, in bits, for the key. Permissible values: `512` or `1024`. You may also enter a custom key size that is between 512 and 1024, in increments of 64 bits. The default value is 1024. Example: `1024`
`exponents`	Limits the possible public exponent values. Use commas to separate different values. Some exponents are more widely used than others. The following exponent values are recommended for arithmetic and security reasons: `17` and `65537`. Of these two values, `65537` is preferred. (This setting is mainly an issue if you are using your own software for generating key pairs. Key-generation programs in Netscape clients and servers use `3` or `65537`.) Permissible values: A combination of `3`, 7, `17`, and `65537`, separated by commas. The default value is `3,7,17,65537`. Example: `17,65537`

DSAKeyRule Rule

The rule named `DSAKeyRule` is an instance of the `DSAKeyConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.
- The minimum key size permitted for certificates is 512 bits (`minSize=512`).
- The maximum key size permitted for certificates is 1024 bits (`maxSize=1024`).
- The exponents allowed are 3, 7, 17, and 65537 (`exponents=3,7,17,65537`).

For details on individual parameters defined in the rule, see Table 17.2 on page 507. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Issuer Constraints Policy

The `IssuerConstraints` plug-in module implements the issuer constraints policy. The policy enables you to effectively deploy certificate-based enrollment explained in "Certificate-Based Enrollment" on page 357.

The policy enables the Certificate Manager to authenticate an end user by checking the issuer DN of the CA that has issued the certificate the user presents as an enrollment token during enrollment. Note that in the current implementation, the CA that issues the new certificates must be the same as the one that has issued the certificates used for SSL client authentication; that is, the issuer DN in the authentication certificate must match the issuer DN specified in the policy configuration.

During installation, Certificate Management System automatically creates an instance of the issuer constraints policy. See “IssuerRule Rule” on page 511. The server also provides appropriate enrollment forms for the three certificate-based enrollment scenarios explained above; see “Enrollment Forms” on page 361.

IssuerConstraints Module

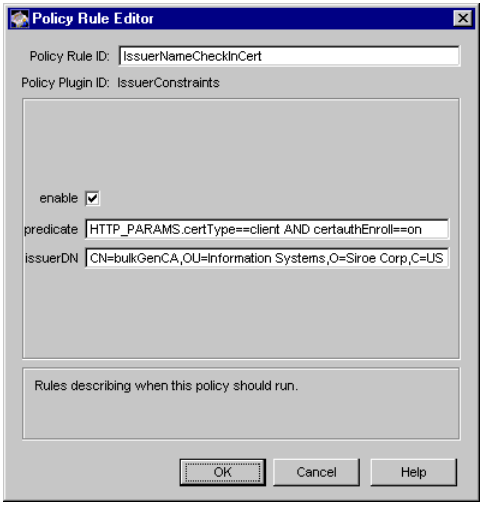
The name of the Java class that implements the issuer constraints policy is as follows:

```
com.netscape.certsrv.policy.IssuerConstraints
```

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.IssuerConstraints.class=com.netscape.certsrv.policy.IssuerConstraints`
- In the CMS window, the module is identified as follows: `IssuerConstraints`

Figure 17.3 shows how the configurable parameters for the `IssuerConstraints` module are displayed in the CMS window.

Figure 17.3 Parameters of the IssuerConstraints module



The configuration shown in Figure 17.3 creates a policy rule named `IssuerNameCheckInCert`, which enforces a rule that the server should check for certificates issued by a CA, whose issuer DN is `CN=bulkGenCA,OU=Information Systems,O=Siroe Corp,C=US`.

Table 17.3 describes each parameter.

Table 17.3 Description of parameters defined in the IssuerConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server checks for certificates issued by the specified CA and enforces certificate-based enrollment. • If you disable the rule, the server does not check for certificates issued by a CA; it ignores the values specified in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client AND certauthEnroll==on`
`issuerDN`	Specifies the name of the CA that has issued certificates that are to be checked. You should enter the issuer name as it appears in the CA's signing certificate; the same name also appears as the issuer name in certificates the CA signs. Permissible values: A valid issuer name. Example: `CN=bulkGenCA,OU=Information Systems,O=Siroe Corp,C=US`

IssuerRule Rule

The rule named `IssuerRule` is an instance of the `IssuerConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.

- The predicate expression is set (`predicate=HTTP_PARAMS.certType==client AND certauthEnroll==on`) so that the rule is applied to only those client-certificate requests that have certificate-based authentication turned on.
- The issuer DN field is left blank for you to enter the appropriate information.

For details on individual parameters defined in the rule, see Table 17.3 on page 510. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Key Algorithm Constraints Policy

The `KeyAlgorithmConstraints` plug-in module implements the key algorithm constraints policy. This policy restricts the key algorithm requested in certificates to the algorithms, such as RSA and DSA, supported by Certificate Management System. In other words, this policy allows you to set restrictions on the types of public keys certified by Certificate Management System.

You may apply this policy to end-entity certificate enrollment and renewal requests. For example, if you want your CA to certify only those public keys that comply with the PKCS-1 RSA Encryption Standard, you can configure the server for that using the policy.

During installation, Certificate Management System automatically creates an instance of the key algorithm constraints policy. See "KeyAlgRule Rule" on page 514.

KeyAlgorithmConstraints Module

The name of the Java class that implements the key algorithm constraints policy is as follows:

```
com.netscape.certsrv.policy.KeyAlgorithmConstraints
```

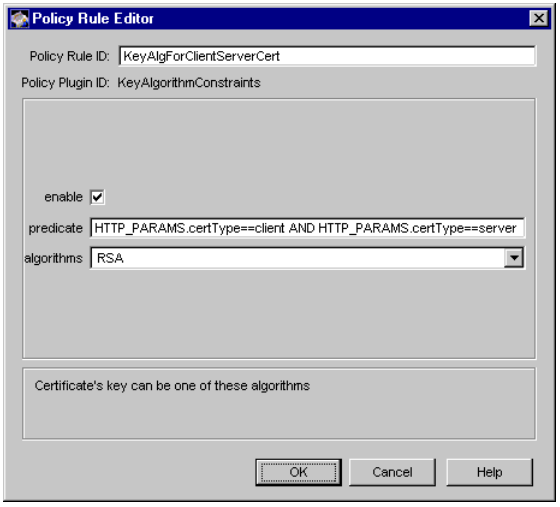
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.KeyAlgorithmConstraints.class=com.netscape.certsrv.policy.KeyAlgorithmConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `KeyAlgorithmConstraints`

Figure 17.4 shows how the configurable parameters for the `KeyAlgorithmConstraints` module are displayed in the CMS window.

Figure 17.4 Parameters of the KeyAlgorithmConstraints module



The configuration shown in Figure 17.4 creates a policy rule named `KeyAlgForClientServerCert`, which enforces a rule that the server should restrict the key algorithm of all client and server certificates to RSA.

Table 17.4 gives details about each of the parameters.

Table 17.4 Description of parameters defined in the KeyAlgorithmConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server sets the configured validity period in renewed certificates specified by the `predicate` parameter. • If you disable the rule, the server sets the validity period as specified in the renewal request.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client AND HTTP_PARAMS.certType==server`
`algorithms`	Specifies the key type the server should certify. The default is RSA. Permissible values: `RSA`, `DSA`, or `RSA,DSA`. Example: `RSA`

KeyAlgRule Rule

The rule named `KeyAlgRule` is an instance of the `KeyAlgorithmConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.
- The key type allowed is RSA (`algorithms=RSA`).

For details on individual parameters defined in the rule, see Table 17.4 on page 513. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

PIN Present Policy

The `PinPresentConstraints` plug-in module implements the PIN present policy. This policy checks for the presence of a user's PIN in the authentication directory and accordingly allows or rejects the certificate request. This policy enables you to effectively deploy one of the directory-based enrollment methods, the password and PIN-based enrollment with PIN removal, by restricting users from enrolling for multiple certificates. For details, see "Setting Up Authentication for End-User Enrollment" on page 387.

In the password and PIN-based enrollment method, users enroll for a certificate using their directory user ID, password and PIN. After a PIN has been used to successfully authenticate a user, the Certificate Manager calls the `PinRemovalListener` module. This module removes the PIN from the authentication directory when the Certificate Manager issues the requested certificate.

Note that listeners in Certificate Management System are objects which register themselves as interested in knowing about certain events—for example, change in the state of a request—and carry out a specific task. For more information on listeners, check the samples directory:

```
<server_root>/cms_sdk/samples/listeners
```

Once the PIN is removed from the authentication directory, it prevents the user from enrolling for another certificate.

The above mentioned process works smoothly if a Certificate Manager or Registration Manager is configured to use the master directory for authenticating users. The process may not work smoothly in deployment

scenarios that involve replicated directories. In these scenarios, you need to use the PIN present policy to verify that the PIN has been removed from the directory. Here's an example of such a scenario:

A Registration Manager acts as an enrollment authority, passing authenticated certificate requests to a Certificate Manager; the users have no direct interaction with the Certificate Manager. The Certificate Manager (CA) and the master corporate directory are behind the firewall. The Registration Manager and a replica of the corporate directory are outside the firewall. The Certificate Manager is configured to communicate with the master corporate directory. The Registration Manager has read-only permission to the replicated corporate directory, and it uses the directory for authenticating end entities. Both the Certificate Manager and Registration Manager are configured for password and PIN-based enrollment with the PIN removal feature turned on. The master corporate directory is configured to update its replica (outside the firewall) every 10 minutes.

When a user enrolls for a certificate request using the End Entity Service interface of the Registration Manager, it authenticates the user against the replica of the corporate directory. If the user presents a valid user ID, password and PIN, the Registration Manager authenticates the user successfully and forwards the request to the Certificate Manager. As the Registration Manager is configured for PIN-based enrollment with PIN removal, it attempts to remove the PIN from the directory, but it can't as it has no write permission to the replicated directory; the PIN is still around.

When the Certificate Manager processes the request forwarded by the Registration Manager, it calls the `PinRemovalListener` module, which in turn removes the PIN from the master corporate directory when the Certificate Manager issues the certificate. (The Certificate Manager sends the certificate to the Registration Manager, which in turn sends it to the user.)

Although the CA has removed the PIN from the master directory, the replicated directory still has the PIN, because the update hasn't occurred. In the meantime, the user may enroll again successfully (from the Registration Manager) for another certificate and receive it from the Certificate Manager.

The PIN present policy enables you to prevent users from successfully enrolling for multiple certificates from the Registration Manager during the time interval between directory updates. If you configure the CA to use this policy to check the master directory for PINs before issuing certificates, successive certificate requests would fail because the PIN has been removed from the master

directory. This way, even if the Registration Manager authenticates successive request, the Certificate Manager rejects them, thus ensuring that a user has only one certificate.

Unlike some of the other policy modules, Certificate Management System does not create an instance of the PIN present policy during installation. If you want a Certificate Manager or Registration Manager to check for PINs, you must create an instance of the `PinPresentConstraints` module and configure it. For instructions, see "Step 3. Enable the PIN Present Policy" on page 393.

PinPresentConstraints Module

The Java class that implements the PIN present policy is as follows:

`com.netscape.certsrv.policy.PinPresentConstraints`

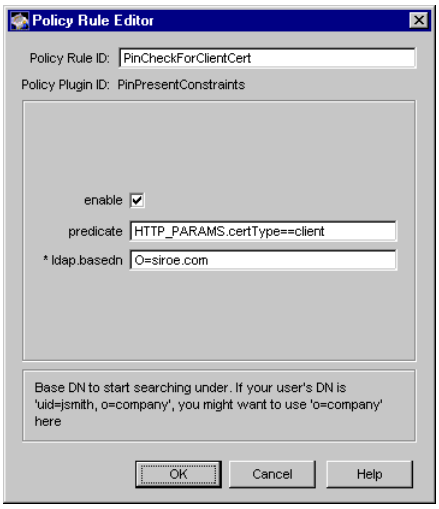
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.PinPresentConstraints.class=com.netscape.certsrv.policy.PinPresentConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `PinPresentConstraints`

Figure 17.5 shows how the configurable parameters for the `PinPresentConstraints` module are displayed in the CMS window.

Figure 17.5 Parameters of the PinPresentConstraints module



The configuration shown in Figure 17.5 creates a policy rule named `PinCheckForClientCert`, which specifies that the server should check the configured authentication directory for PINs before issuing a client certificate. To locate the user's PIN attribute in the directory, the server should start the search from the `O=siroe.com` node.

Table 17.5 gives details about each of the parameters.

Table 17.5 Description of parameters defined in the PinPresentConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server checks the configured authentication directory for users' PINs and accordingly allows or denies certificate requests. • If you disable the rule, the server does not check the authentication directory for PINs; upon successful authentication, it issues the certificate.

Table 17.5 Description of parameters defined in the PinPresentConstraints module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`ldap.basedn`	Specifies the base DN for searching the authentication directory—the server uses the user ID from the HTTP input (what a user enters in the enrollment from) and the base DN to construct an LDAP search filter. Note that the base DN you specify here must match the one specified in the authentication rule. Permissible values: Any valid DN string of up to 255 characters. Example: `O=siroe.com`

Renewal Constraints Policy

The `RenewalConstraints` plug-in module implements the renewal constraints policy. This policy imposes constraints on renewal of expired certificates—it allows or restricts the server from renewing expired certificates. You may apply this policy to end-entity certificate renewal requests. For example, if you don't want to allow renewal of expired certificates, you can configure the server accordingly using the policy.

In certain situations you may want to allow renewal of expired certificates. Here's one such scenario: the renewal validity constraints policy (see "Renewal Validity Constraints Policy" on page 525) allows you to delay renewal of certificates as long as possible to reduce the overhead of processing new certificate requests. Typically, you would limit the renewal process to the last few weeks of validity of the certificate. However, if the interval specified in the policy rule is not sufficient for renewal to occur, some of your users may not be able to renew their certificates prior to the expiration time and end up owning expired certificates.

Note that the policy also allows you to specify how long after the expiration of a certificate can it be renewed. If you don't specify this, the server will renew all expired certificates that are submitted for renewal.

During installation, Certificate Management System automatically creates an instance of the renewal constraints policy. See "RenewalConstraintsRule Rule" on page 521.

RenewalConstraints Module

The Java class that implements the renewal constraints policy is as follows:

`com.netscape.certsrv.policy.RenewalConstraints`

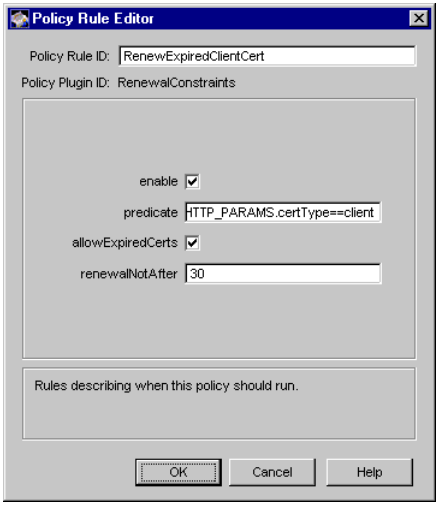
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.RenewalConstraints.class=com.netscape.certsrv.policy.RenewalConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `RenewalConstraints`

Figure 17.6 shows how the configurable parameters for the `RenewalConstraints` module are displayed in the CMS window.

Figure 17.6 Parameters of the RenewalConstraints module



The configuration shown in Figure 17.6 creates a policy rule named `RenewExpiredClientCert`, which specifies that the server should allow renewal of expired client certificates, if it's done within 30 days from the expiry date.

Table 17.6 gives details about each of the parameters.

Table 17.6 Description of parameters defined in the RenewalConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server verifies the validity period of the certificate being renewed, checks the value assigned to the `allowExpiredCerts` parameter, and accordingly allows or denies the renewal request. • If you disable the rule, the server does not verify the validity period of the certificate being renewed; it simply renews the certificate.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`allowExpiredCerts`	Specifies whether to allow or prevent renewal of expired certificates. Check the box if you want the server to renew expired certificates (default). Uncheck the box if you don't want the server to renew expired certificates.
`renewalNotAfter`	Specifies how long, in days, after the expiration of a certificate can it be renewed. The default value is 30 days. If you leave the field blank, the server will renew all expired certificates that are submitted for renewal. Example: 15

RenewalConstraintsRule Rule

The rule named `RenewalConstraintsRule` is an instance of the `RenewalConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the policy is applied to all certificate renewal requests processed by the server.
- The server allows renewal of expired certificates within 30 days, starting from the date they expire.

For details on individual parameters defined in the rule, see Table 17.6 on page 520. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Revocation Constraints Policy

The `RevocationConstraints` plug-in module implements the revocation constraints policy. This policy imposes constraints on revocation of expired certificates—it allows or restricts the server from revoking expired certificates. You may apply this policy to end-entity certificate revocation requests. For example, if you don't want to allow revocation of expired certificates in your PKI setup, you can configure the server accordingly using the policy.

During installation, Certificate Management System automatically creates an instance of the revocation constraints policy. See "RevocationConstraintsRule Rule" on page 524.

RevocationConstraints Module

The Java class that implements the revocation constraints policy is as follows:

`com.netscape.certsrv.policy.RevocationConstraints`

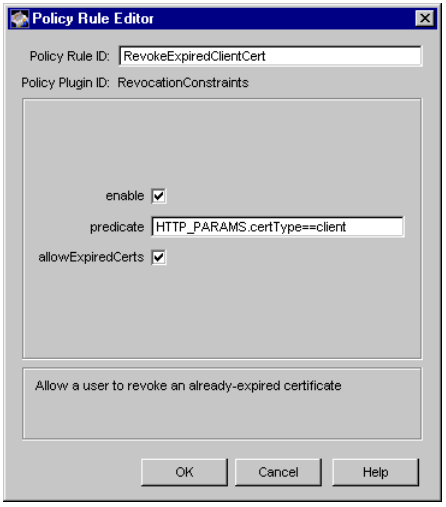
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.RevocationConstraints.class=com.netscape.certsrv.policy.RevocationConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `RevocationConstraints`

Figure 17.7 shows how the configurable parameters for the `RevocationConstraints` module are displayed in the CMS window.

Figure 17.7 Parameters of the RevocationConstraints module



The configuration shown in Figure 17.7 creates a policy rule named `RevokeExpiredClientCert`, which specifies that the server should allow revocation of expired client certificates.

Table 17.7 gives details about each of the parameters.

Table 17.7 Description of parameters defined in the RevocationConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server verifies the validity period of the certificate being revoked, checks the value assigned to the `allowExpiredCerts` parameter, and accordingly allows or denies the revocation request. • If you disable the rule, the server does not verify the validity period of the certificate being revoked; it simply revokes the certificate.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`allowExpiredCerts`	Specifies whether to allow or prevent revocation of expired certificates. Check the box if you want the server to revoke expired certificates (default). Uncheck the box if you don't want the server to revoke expired certificates.

RevocationConstraintsRule Rule

The rule named `RevocationConstraintsRule` is an instance of the `RevocationConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the policy is applied to all certificate revocation requests processed by the server.
- The server allows revocation of expired certificates.

For details on individual parameters defined in the rule, see Table 17.7 on page 523. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Renewal Validity Constraints Policy

The `RenewalValidityConstraints` plug-in module implements the renewal validity constraints policy. This policy governs the formulation of content in the renewed certificate based on the currently issued certificate.

Every certificate issued by Certificate Management System is valid for a limited duration, which is determined by the validity period specified in the validity constraints policy (see "Validity Constraints Policy" on page 543) at the time the certificate is issued. In order to continue to participate in the PKI-using system beyond this validity period, the entity owning the certificate must renew the certificate; the new certificate generally contains a new validity time period and some updated attributes.

To eliminate administrative overhead of monitoring certificate validity periods and reminding users to renew their certificates, Certificate Management System provides a schedulable job that can detect any to-be-expired certificates and automatically remind users to renew their certificates; for details about this job, see "Certificate Renewal Notifications" on page 435.

The renewal validity constraints policy enables you to enforce certain restrictions on certificate-renewal requests, when end entities attempt to renew their certificates. You can specify restrictions on the following:

- The number of days before expiration that end entities can renew their currently active or valid certificates. For example, if you want to prevent end entities from renewing their certificates any earlier than 15 days before expiration, you can configure the server accordingly using the policy.
- The validity period of the renewed certificate. For example, if you want the validity period of all renewed certificates to be a minimum of 180 days, you can configure the server accordingly using the policy. Note that the renewal period starts from the ending period in the certificate presented for renewal.

Note that you may apply this policy to certificate renewal requests only, and the renewal process to which this policy is applied can be manual (a request needs to be approved by an agent) or automated; for details, see "Certificate Renewal" on page 1111. In both cases, the currently issued certificate must be either presented during SSL client authentication by the end entity or selected by the agent approving the renewal request.

By default, any validity requested in a certificate-renewal request cannot exceed beyond that of the expiration time specified in the CA's signing certificate (see "CA Signing Key Pair and Certificate" on page 226). If the Certificate Manager (CA) finds a request with validity period extending beyond that of its CA signing certificate, it automatically truncates the validity period to end on the day the CA signing certificate expires. For example, if the CA signing certificate expires on June 10, 2004, any renewal request with validity period beyond June 10, 2004 will have validity period truncated to end on June 10, 2004.

However, you can configure the Certificate Manager to renew certificates with validity periods beyond that of its CA signing certificate by selecting the "Override validity nesting requirement" option; see "Step 6. Enable End-Entity Interaction" on page 405.

During installation, Certificate Management System automatically creates an instance of the renewal validity constraints policy. See "DefaultRenewalValidityRule Rule" on page 528.

RenewalValidityConstraints Module

The Java class that implements the renewal validity constraints policy is as follows:

```
com.netscape.certsrv.policy.RenewalValidityConstraints
```

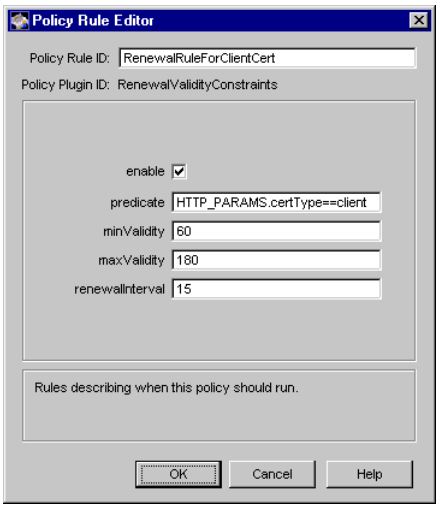
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.RenewalValidityConstraints.class=`
  `com.netscape.certsrv.policy.RenewalValidityConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `RenewalValidityConstraints`

Figure 17.8 shows how the configurable parameters for the `RenewalValityConstraints` module are displayed in the CMS window.

Figure 17.8 Parameters of the RenewalValidityConstraints module



The configuration shown in Figure 17.8 creates a policy rule named `RenewalRuleForClientCert`, which enforces a rule that the server should renew only those client certificates that are due to expire within the next 15 days. The renewed certificates are valid for at least 60 days (two months) and require renewing after 180 days (six months).

Table 17.8 gives details about each of the parameters.

Table 17.8 Description of parameters defined in the RenewalValidityConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server sets the configured validity period in renewed certificates specified by the `predicate` parameter. • If you disable the rule, the server sets the validity period as specified in the renewal request.

Table 17.8 Description of parameters defined in the RenewalValidityConstraints module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`minValidity`	Specifies the minimum validity period, in days, for renewed certificates. Permissible values: As applicable. The default value is 180 days. Example: `60`
`maxValidity`	Specifies the maximum validity period, in days, for renewed certificates. Permissible values: As applicable. The default value is 730 days. Example: `180`
`renewalInterval`	Specifies how many days before its expiration that a certificate can be renewed. Permissible values: As applicable. The default value is 15 days. Example: `15`

DefaultRenewalValidityRule Rule

The rule named `DefaultRenewalValidityRule` is an instance of the `RenewalValidityConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the policy is applied to all certificate renewal requests processed by the server.
- The minimum validity period permitted for renewed certificates is 30 days (`minValidity=30`).

- The maximum validity period permitted for renewed certificates is 365 days (`maxValidity=365`).
- The number of days before expiration that end entities can renew their currently valid certificates is 15 (`renewalInterval=15`).

For details on individual parameters defined in the rule, see Table 17.8 on page 527. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

RSA Key Constraints Policy

The `RSAKeyConstraints` plug-in module implements the RSA key constraints policy. This policy imposes constraints on the following:

- The minimum and maximum sizes for keys
- The exponent sizes

The policy restricts the key size to one of the sizes supported by Certificate Management System—512, 1024, 2048, or 4096. In other words, the policy allows you to set up restrictions on the lengths of public keys certified by Certificate Management System.

You may apply this policy to end-entity certificate enrollment and renewal requests. For example, if you want your CA to certify public keys up to 1024 bits in length for end users, you can configure the server accordingly using the policy.

During installation, Certificate Management System automatically creates an instance of the RSA key constraints policy. See "RSAKeyRule Rule" on page 532.

RSAKeyConstraints Module

The Java class that implements the RSA key constraints policy is as follows:

`com.netscape.certsrv.policy.RSAKeyConstraints`

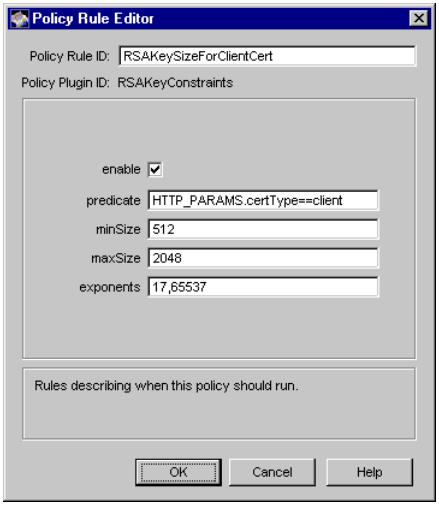
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.RSAKeyConstraints.class=com.netscape.certsrv.policy.RSAKeyConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `RSAKeyConstraints`

Figure 17.9 shows how the configurable parameters for the `RSAKeyConstraints` module are displayed in the CMS window.

Figure 17.9 Parameters of the RSAKeyConstraints module



The configuration shown in Figure 17.9 creates a policy rule named `RSAKeySizeForClientCert`, which enforces a rule that the server should restrict the minimum and maximum key sizes for all RSA key-based client certificates to 512 and 2048, respectively.

Table 17.9 describes each parameter.

Table 17.9 Description of parameters defined in the RSAKeyConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server uses the configured RSA key rules when issuing certificates specified by the `predicate` parameter. • If you disable the rule, the server certifies the requested key size.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`minSize`	Specifies the minimum length, in bits, for the key (the length of the modulus in bits). The value must be smaller than or equal to the one specified by the `maxSize` parameter. In general, a longer key size results in a key pair that is more difficult to crack. You may want to allow a minimum length to ensure a minimum level of security. Permissible values: `512`, `1024`, `2048`, or `4096`. You may also enter a custom key size that is between 512 and 4096 bits. The default value is 512. Example: `512`
`maxSize`	Specifies the maximum length, in bits, for the key. Permissible values: `512`, `1024`, `2048`, or `4096`. You may also enter a custom key size that is between 512 and 4096 bits. The default value is 2048. Example: `1024`

Table 17.9 Description of parameters defined in the RSAKeyConstraints module (Continued)

Parameter	Description
`exponents`	Limits the possible public exponent values. Use commas to separate different values. Some exponents are more widely used than others. The following exponent values are recommended for arithmetic and security reasons: `17` and `65537`. Of these two values, `65537` is preferred. (This setting is mainly an issue if you are using your own software for generating key pairs. Key-generation programs in Netscape clients and servers use `3` or `65537`.) Permissible values: A combination of `3`, 7, `17`, and `65537`, separated by commas. The default value is `3,7,17,65537`. Example: `17,65537`

RSAKeyRule Rule

The rule named `RSAKeyRule` is an instance of the `RSAKeyConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.
- The minimum key size permitted for certificates is 512 bits (`minSize=512`).
- The maximum key size permitted for certificates is 2048 bits (`maxSize=2048`).
- The exponents allowed are 3, 7, 17, and 65537 (`exponents=3,7,17,65537`).

For details on individual parameters defined in the rule, see Table 17.9 on page 531. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Signing Algorithm Constraints Policy

The `SigningAlgorithmConstraints` plug-in module implements the signing algorithm constraints policy. This policy restricts the requested signing algorithm to be one of the algorithms supported by Certificate Management System: MD2 with RSA, MD5 with RSA, and SHA-1 with RSA, if the Certificate Manager's signing key is RSA and SHA-1 with DSA, if the Certificate Manager's signing key is DSA.

When a Certificate Manager digitally signs a message, it generates a compressed version of the message called a message digest. Some of the algorithms used to produce this digest include MD5 and SHA-1 (Secure Hash Algorithm).

- MD5 generates a 128-bit message digest. Most existing software applications that handle certificates only support MD5.
- SHA-1 generates a 160-bit message digest. Some software applications do not yet support the SHA-1 algorithm. For example, Netscape Navigator 3.0 (or higher) and Enterprise Server 2.01 (or higher) support SHA-1; previous versions of these applications do not support SHA-1.

You may apply this policy to end-entity certificate enrollment and renewal requests.

During installation, Certificate Management System automatically creates an instance of the signing algorithm constraints policy. See "SigningAlgRule Rule" on page 536.

SigningAlgorithmConstraints Module

The Java class that implements the signing algorithm constraints policy is as follows:

```
com.netscape.certsrv.policy.SigningAlgorithmConstraints
```

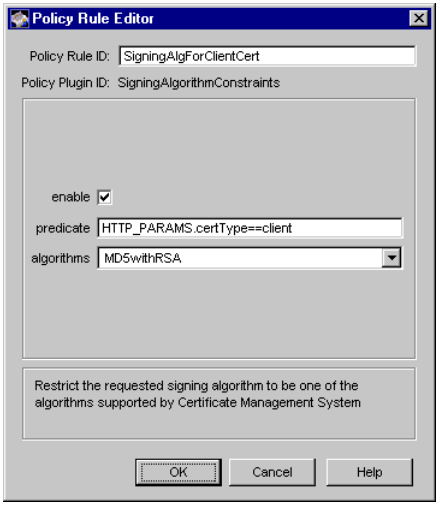
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.SigningAlgorithmConstraints.class=`
  `com.netscape.certsrv.policy.SigningAlgorithmConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `SigningAlgorithmConstraints`

Figure 17.10 shows how the configurable parameters for the `SigningAlgorithmConstraints` module are displayed in the CMS window.

Figure 17.10 Parameters of the SigningAlgorithmConstraints module



The configuration shown in Figure 17.10 creates a policy rule named `SigningAlgForClientCert`, which enforces a rule that the server should use MD5 with RSA signature algorithm to sign all client certificates.

Table 17.10 provides details for each of these parameters.

Table 17.10 Description of parameters defined in the SigningAlgorithmConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server uses the configured algorithms to sign certificates specified by the `predicate` parameter. • If you disable the rule, the server uses the default algorithm specified in the General Settings tab; see "Enabling End-Entity Interaction with a Certificate Manager" on page 405.
`predicate`	Specifies the predicate expression for this rule. If you want the rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`algorithms`	Specifies the signature algorithm the server should use to sign certificates. Permissible values: Depends on the CA's signing key type (the key type you chose for the Certificate Manager's CA signing certificate). • If the key type is RSA, select one of the following: - `MD2withRSA,MD5withRSA,SHA1withRSA` - `MD2withRSA,MD5withRSA` - `MD2withRSA,SHA1withRSA` - `MD5withRSA,SHA1withRSA` - `MD2withRSA` - `MD5withRSA` - `SHA1withRSA` The default value is `MD2withRSA,MD5withRSA,SHA1withRSA`. • If the key type is DSA, select `SHA1withDSA`. Example: `MD5withRSA`

SigningAlgRule Rule

The rule named `SigningAlgRule` is an instance of the `SigningAlgorithmConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.
- The signature algorithms allowed are MD5 with RSA, MD2 with RSA, and SHA-1 with RSA (`algorithms=MD5withRSA,MD2withRSA,SHA1withRSA`).

For details on individual parameters defined in the rule, see Table 17.10 on page 535. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Subordinate CA Name Constraints Policy

The `SubCANameConstraints` plug-in module implements the subordinate CA name constraints policy. This policy restricts a CA from issuing a subordinate CA certificate that has the same issuer name as that of the CA itself—that is, the policy prevents a situation where the signing certificates of a CA and its subordinate CA have identical issuer names.

This policy must be turned on if you're planning to issue subordinate CA certificates. The reason for this is that, whenever the Certificate Manager issues a certificate, it stores the related information in its internal database; see "Internal Database" on page 163. If the CA issues a subordinate CA certificate with an issuer DN that matches its own issuer DN, the internal database will not function properly.

You may apply this policy to CA certificate enrollment and renewal requests.

During installation, Certificate Management System automatically creates an instance of the subordinate CA name constraints policy. See "SubCANameConstraints Rule" on page 538.

SubCANameConstraints Module

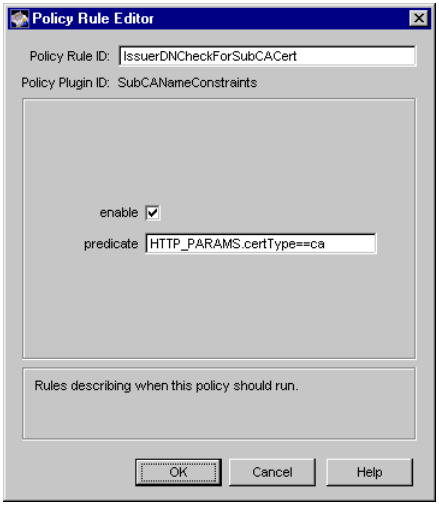
The Java class that implements the subordinate CA name constraints policy is as follows:

`com.netscape.certsrv.policy.SubCANameConstraints`

- In the CMS configuration file, the module is identified as follows:
  `ca.Policy.impl.SubCANameConstraints.class=com.netscape.certsrv.policy.SubCANameConstraints`
- In the CMS window, the module is identified as follows:
  `SubCANameConstraints`

Figure 17.11 shows how configurable parameters for the `SubCANameConstraints` module are displayed in the CMS window.

Figure 17.11 Parameters of the SubCANameConstraints module



The configuration shown in Figure 17.11 creates a policy rule named `IssuerDNCheckForSubCACert`, which enforces a rule that the server should reject subordinate CA certificate requests with issuer DNs matching its own issuer DN.

Table 17.11 gives details about each of the parameters.

Table 17.11 Description of parameters defined in the SubCANameConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server checks the certificate requests for issuer name uniqueness. If a certificate with the requested issuer name already exists in the internal database, the server rejects the request. • If you disable the rule, the server does not check the CA certificate requests for issuer name uniqueness.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`

SubCANameConstraints Rule

The rule named `SubCANameConstraints` is an instance of the `SubCANameConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.

For details on individual parameters defined in the rule, see Table 17.12 on page 541. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Unique Subject Name Constraints Policy

The `UniqueSubjectNameConstraints` plug-in module implements the unique subject name constraints policy. This policy restricts the server from issuing multiple certificates with same subject names. Optionally, you can also configure the server to allow multiple certificates with the same subject name if the key usages are different. Note that key usages for certificates are usually specified by the *key usage extension* and Certificate Management System allows you to add this extension to certificates using the key usage extension policy explained in "Key Usage Extension Policy" on page 618.

You may apply the unique subject name constraints policy to end-entity certificate enrollment and renewal requests. For example, if you want to prevent your users from requesting multiple certificates with same subject names, you can configure the server accordingly using the policy. Alternatively, if you want to allow your users to own multiple certificates, each for a different use, all having the same subject name, you can do so easily using the `enableKeyUsageExtensionChecking` parameter defined in this policy. This parameter makes the server check whether the key usages specified in the certificate request being processed is different than those specified in the existing certificates that have the same subject names and accordingly issue or deny the certificate. Keep in mind that the server can check for key usages only if the key usage extension bits are set in the certificate request being processed as well as in the existing certificates that have the same subject names.

During installation, Certificate Management System automatically creates an instance of the unique subject name constraints policy. See "UniqueSubjectNameConstraints Rule" on page 542.

UniqueSubjectNameConstraints Module

The Java class that implements the unique subject name constraints policy is as follows:

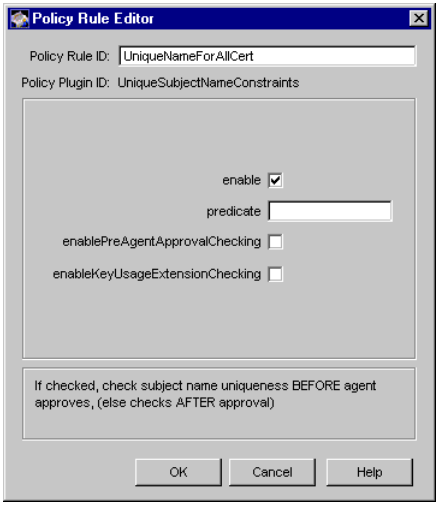
```
com.netscape.certsrv.policy.UniqueSubjectNameConstraints
```

- In the CMS configuration file, the module is identified as follows:
  `ca.Policy.impl.UniqueSubjectNameConstraints.class=com.netscape.certsrv.policy.UniqueSubjectNameConstraints`

- In the CMS window, the module is identified as follows: `UniqueSubjectNameConstraints`

Figure 17.12 shows how configurable parameters for the `UniqueSubjectNameConstraints` module are displayed in the CMS window.

Figure 17.12 Parameters of the UniqueSubjectNameConstraints module



The configuration shown in Figure 17.12 creates a policy rule named `UniqueNameForAllCert`, which enforces a rule that all certificates must have unique subject names.

Table 17.12 describes each of the parameters.

Table 17.12 Description of parameters defined in the UniqueSubjectNameConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server checks the certificate requests, as specified by the `predicate` parameter, for subject name uniqueness. • If you disable the rule, the server does not check the certificate requests for subject name uniqueness.

Table 17.12 Description of parameters defined in the UniqueSubjectNameConstraints module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`enablePreAgentApprovalChecking`	Specifies whether the request must be checked for the subject name uniqueness on submission by the user, before the request gets queued for agent approval. • Check the box if you want the server to check the certificate request for the subject name uniqueness as soon as the user submits it. • Uncheck the box if you want the server to check the certificate request for the subject name uniqueness after agent approval; that is, you want the policy to be applied to the request after an agent approves the request. You should choose this option if you want the server to check the Key Usage extension (see "Key Usage Extension Policy" on page 618) before determining whether to issue the certificate.
`enableKeyUsageExtensionChecking`	Specifies whether the certificate request must be checked for the Key Usage extension (see "Key Usage Extension Policy" on page 618). Note that the policy can check the certificate request for the Key Usage extension only if you uncheck (disable) the `enablePreAgentApprovalChecking` parameter. The reason for this is that, extensions are set on the request after agent approval, so this checking can be done after an agent approves the request. • Check the box if you want the server to check the certificate request for the Key Usage extension. If you check the box, the server checks its internal database for certificates that have the same subject name as the one specified in the request. For each certificate that has the matching subject name, the server compares the Key Usage extension of the certificate to the one specified in the request. If the server finds a certificate that has the same subject name and Key Usage extension, it rejects request. Otherwise, the server approves the request. (This choice is suitable if you want to have multiple certificates with same subject names but for different purposes, such as signing and encrypting. If key-usage comparison is to be done, be sure to specify that this policy is to be applied after the Key Usage extension policy; see "Step 5. Reorder Policy Rules" on page 705.) • Uncheck the box if you don't want the server to check the certificate request for the Key Usage extension. If you uncheck the box, the server does not compare the Key Usage extension in the request with the ones set in the existing certificates that have the same subject name; it simply rejects requests with same subject names.

UniqueSubjectNameConstraints Rule

The rule named `UniqueSubjectNameConstraints` is an instance of the `UniqueSubjectNameConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The certificate requests are checked for subject name uniqueness after agents process the requests for approval—if you're using manual enrollment and *deferred* requests.
- The certificate requests are checked for Key Usage extension.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.

For details on individual parameters defined in the rule, see Table 17.12 on page 541. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Validity Constraints Policy

The `ValidityConstraints` plug-in module implements the validity constraints policy. This policy enforces minimum and maximum validity periods for certificates and changes them if the policy is not met. Specifically, the policy imposes constraints on the following:

- The duration of a certificate's validity period (based on supported minimum and maximum validity periods).
- The lead and lag time for the beginning date and time (the `notBefore` and `notAfter` attributes in certificate requests) for the validity period; how far back into the front or back the `notBefore` date could go in minutes.

If this policy rule is enabled, the server applies the rule to the certificate request being processed, and then determines if the validity period in the request is acceptable. The rule checks two X.509 attributes of the certificate, the `notBefore` and `notAfter` time, which together indicate the total validity life of a certificate, to make sure that they conform to the configured ranges.

The rule checks that the value of the `notBefore` attribute in the request is not more than `leadTime` minutes in the future; the `leadTime` is a configurable parameter in the plug-in implementation. The ability to configure the value of the `leadTime` parameter in the policy rule allows you to prohibit end entities from requesting certificates whose validity starts too far in the future, and yet allows some amount of toleration of clock-skew problems. For example, if the current date and time is 01/15/2000 (mm/dd/YYYY) and 1:30 p.m., the value of the `notBefore` attribute is set to 3:00 p.m., and that the `leadTime` is 10 minutes, then the request would fail, because the validity requested begins more than 10 minutes in the future.

The rule also checks that the value of the `notBefore` attribute in the request is not more than `lagTime` minutes in the past. For example, if the current date and time is 01/15/2000 (mm/dd/yyyy) and 1:30 p.m., the value of the `notBefore` attribute is set to 1:15 p.m., and the `lagTime` is set to 10 minutes, the request would fail because the user has requested a certificate 15 minutes in the past. Note that a request with `notBefore` set to 1:25 p.m. would have passed, however.

Note When applying the validity constraints policy, the server does not check the lag time in all certificate requests. It checks the lag time only in those requests that are based on the CRMF protocol—currently, CRMF is the only enrollment format that allows an end entity to request a specific validity period with the `notBefore` attribute set to a time in the past.

You may apply this policy to end-entity certificate enrollment requests. It can be useful to restrict the length of the validity period for certificates issued by the server. For example, if you want users to renew their certificates at least once a year, you can set the maximum validity period to one year. If you want to limit the frequency of certificate renewals to keep down administrative costs, you can set the minimum validity period to six months.

By default, any validity requested in a certificate enrollment request cannot exceed beyond that of the expiration time specified in the CA's signing certificate (see "CA Signing Key Pair and Certificate" on page 226). If the Certificate Manager (CA) finds a request with validity period extending beyond that of its CA signing certificate, it automatically truncates the validity period to

end on the day the CA signing certificate expires. For example, if the CA signing certificate expires on June 10, 2004, any enrollment request with validity period beyond June 10, 2004 will have validity period truncated to end on June 10, 2004.

However, you can configure the Certificate Manager to issue certificates with validity periods beyond that of its CA signing certificate by selecting the "Override validity nesting requirement" option; see "Step 6. Enable End-Entity Interaction" on page 405.

During installation, Certificate Management System automatically creates an instance of the validity constraints policy. See "DefaultValidityRule Rule" on page 548.

ValidityConstraints Module

The Java class that implements the validity constraints policy is as follows:

`com.netscape.certsrv.policy.ValidityConstraints`

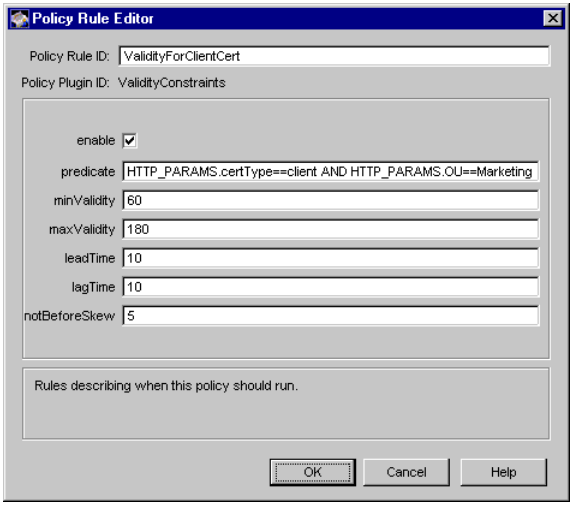
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.ValidityConstraints.class=com.netscape.certsrv.policy.ValidityConstraints`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `ValidityConstraints`

Figure 17.13 shows how configurable parameters for the `ValidityConstraints` module are displayed in the CMS window.

Figure 17.13 Parameters of the ValidityConstraints module



The configuration shown in Figure 17.13 creates a policy rule named `ValidityForClientCert`, which enforces a rule that all client certificates requested by end users in an organizational unit (`OU`) called Marketing are valid for at least 60 days (two months) and require renewing after 180 days (six months).

Table 17.13 gives details about each of the parameters.

Table 17.13 Description of parameters defined in the ValidityConstraints module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server sets the configured validity period in certificates specified by the `predicate` parameter. • If you disable the rule, the server does not set the configured validity period in certificates; it sets the validity period to the one specified in the request.

Table 17.13 Description of parameters defined in the ValidityConstraints module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client AND HTTP_PARAMS.OU==Marketing`
`minValidity`	Specifies the minimum validity period, in days, for certificates. Permissible values: An integer greater than zero and less than the value specified by the `maxValidity` parameter. The default value is 180 days. Example: `60`
`maxValidity`	Specifies the maximum validity period, in days, for certificates. Permissible values: An integer greater than zero and also greater than the value specified by the `minValidity` parameter. The default value is 730 days. Example: `180`
`leadTime`	Specifies the lead time, in minutes, for certificates. For a certificate renewal request to pass the renewal validity constraints policy, the value of the `notBefore` attribute in the certificate request must not be more than value of the `leadTime` parameter in the future, relative to the time when the policy rule is run. The `notBefore` attribute value specifies the date on which the certificate validity begins; validity dates through the year 2049 are encoded as UTCTime, dates in 2050 or later are encoded as GeneralizedTime. Permissible values: As applicable. The default value is 10 minutes. Example: `10`

Table 17.13 Description of parameters defined in the ValidityConstraints module (Continued)

Parameter	Description
`lagTime`	Specifies the lag time, in minutes, for certificates. For a certificate renewal request to pass the renewal validity constraints policy, the value of the `notBefore` attribute in the certificate request must not be more than the value of the `lagTime` in the past, relative to the time when the policy is run. The `notBefore` attribute value specifies the date on which the certificate validity ends; validity dates through the year 2049 are encoded as UTCTime, dates in 2050 or later are encoded as GeneralizedTime. Permissible values: As applicable. The default value is 10 minutes. Example: `10`
`notBeforeSkew`	Specifies the number of minutes to subtract from the current time when creating the value for the certificate's `notBefore` attribute. It can help some clients with incorrectly set clocks use the new certificate after downloading. For example, if the certificate is issued at 11:30 a.m. and the clock settings of the client into which the certificate is downloaded is 11:20 a.m., the certificate cannot be used for 10 minutes. Setting the value of the `beforeFix` parameter to 10 minutes would adjust the value of the `notBefore` parameter to 11:20 a.m.—thus making the certificate usable following the download. Permissible values: As applicable. The default value is 5 minutes. Example: `5`

DefaultValidityRule Rule

The rule named `DefaultValidityRule` is an instance of the `ValidityConstraints` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the rule is applied to all certificate enrollment and renewal requests processed by the server.
- The minimum validity period allowed for certificates is 30 days (`minValidity=30`).

- The maximum validity period allowed for certificates is 365 days (`maxValidity=365`).
- The lead time allowed is 10 minutes (`leadTime=10`).
- The lag time allowed is 10 minutes (`lagTime=10`).
- The the number of minutes to subtract from the current time when creating the value for the certificate's `notBefore` attribute is 5 minutes (`notBeforeSkew=5`).

For details on individual parameters defined in the rule, see Table 17.13 on page 546. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Chapter

18

Extension-Specific Policy Modules

Certificate Management System comes with a set of extension-specific policy plug-in modules that enable you to add X.509 certificate extensions to the certificates the server issues. This chapter explains those modules—it lists and briefly describes the modules that are installed with a Certificate Manager and Registration Manager, and then explains each one in detail.

The chapter has the following sections:

Certificate Extensions

Since its initial publication, the X.509 standard for certificate formats has been amended to include additional information within a certificate. Version 3, the latest version, allows you to add additional information as certificate extensions. For example, Netscape applications (Netscape Navigator 3.0 or higher, and Enterprise Server 2.01 or higher) support an extension that specifies the type of certificate issued (such as client, server, or object signing).

The extensions defined for X.509 v3 certificates enable you to associate additional attributes with users or public keys and manage the certification hierarchy. The *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* (see `http://www.ietf.org/rfc/rfc2459.txt`) recommends a set of extensions to be used in Internet certificates (and standard locations for certificate or CA information). These extensions are called *standard extensions*.

The standard also suggests that you can define your own extensions and include them in certificates you issue. These extensions are called *private*, *proprietary*, or *custom* extensions and they carry information unique to your organization or business. Keep in mind that applications may not able to validate certificates that contain private, critical extensions, thus preventing the use of these certificates in a general context.

Note Some explanations in this chapter make reference to Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER). These are specified in the CCITT Recommendations X.208 and X.209. For a quick summary of ASN.1 and DER, see *A Layman's Guide to a Subset of ASN.1, BER, and DER*, which is available at RSA Laboratories' web site (`http://www.rsa.com`).

Structure of Certificate Extensions

In RFC 2459, an X.509 certificate extension is defined as follows:

```
Extension  ::=  SEQUENCE  {
   extnID      OBJECT IDENTIFIER,
   critical    BOOLEAN DEFAULT FALSE,
   extnValue   OCTET STRING  }
```

Which means, a certificate extension consists of the following:

- The object identifier (OID) for the extension; see "Object Identifier" on page 553.

  This identifier uniquely identifies the extension. It also determines the ASN.1 type of value in the value field and how the value is interpreted. That is, when an extension appears in a certificate, the OID appears as the extension ID field (`extnID`) and the corresponding ASN.1 encoded structure appears as the value of the octet string (`extnValue`); see the examples in "Sample Certificate Extensions" on page 552.

- A flag or boolean field called `critical`.

  The value, which can be either true or false, assigned to this field indicates whether the extension is critical or noncritical to the certificate.

  — If the extension is critical and the certificate is sent to an application that does not understand the extension (based on the extension's ID), the application must reject the certificate.

— If the extension is not critical and the certificate is sent to an application that does not understand the extension (based on the extension's ID), the application can ignore the extension and accept the certificate.

- An octet string containing the DER encoding of the value of the extension.

  Typically, the application receiving the certificate checks the extension ID to determine if it can recognize the ID. If it can, it uses the extension ID to determine the type of value used.

Examples of standard extensions defined in the X.509 v3 standard include the following:

- Authority Key Identifier Extension—an extension for identifying the certificate authority's public key (the key used to sign the certificate).
- Subject Key Identifier Extension—an extension for identifying the subject's public key (the key being certified).

Note that not all applications support certificates with version 3 extensions. Applications that do support these extensions may not be able to interpret some or all of these specific extensions.

Sample Certificate Extensions

The following is an example of the section of a certificate containing X.509 v3 extensions. (Certificate Management System can display certificates in human-readable format, as shown here.) As shown in the example, certificate extensions appear in sequence and only one instance of a particular extension may appear in a particular certificate; for example, a certificate may contain only one subject key identifier extension. Note that certificates that support these extensions have the version 0x2 (which corresponds to version 3).

```
Certificate:
   Data:
      Version: v3 (0x2)
...
Extensions:
   Identifier: Certificate Type
      Critical: no
      Certified Usage:
         SSL CA
   Identifier: Subject Key Identifier
      Critical: no
```

```
      Value:
      2c:22:c6:ae:4e:4b:91:c7:fb:4c:cc:ae:84:e8:aa:5b:46:6a:a0:ad
   Identifier: Authority Key Identifier
      Critical: no
      Key Identifier:
      2c:22:c6:ae:4e:4b:91:c7:fb:4c:cc:ae:84:e8:aa:5b:46:6a:a0:ad
```

Object Identifier

An object identifier (OID) is a string of numbers identifying a unique object, for example, a certificate extension or a company's certificate practice statement. For general information on OIDs, see the information at this URL:

`http://www.alvestrand.no/objectid/`

OIDs are controlled by the International Standards Organization (ISO) registration authority. In some cases, this authority is delegated by ISO to regional registration authorities. For example, in the United States, the American National Standards Institute (ANSI) manages this registration.

Registration of Object Identifiers

To promote interoperatability, the PKIX standard recommends that all objects (such as extensions and policy statements) that appear in certificates that will be used in networks shared by other organizations should be included in the form of OIDs. If you plan to issue certificates that will be used in such networks, you should register your object identifier prefixes with the appropriate registration authority. For example, assume you want to add a custom extension that points to a certificate practice statement (CPS) of your company. To implement this, you need to compose the policy statement you want to include in the extension, define an OID for the policy statement, and configure Certificate Management System with the OID so that it can add that to the certificate it issues.

The use of an OID registered to another organization or the failure to register an OID may carry legal consequences, depending on context. Registration may be subject to fees. For more information, you should contact the appropriate registration authority.

To define or assign OIDs for your objects, you must know your company's *arc*, which is an OID for a private enterprise. If your company doesn't have an arc, it needs to get one. This URL contains information on registering for a company arc:

```
http://www.isi.edu/cgi-bin/iana/enterprise.pl
```

To understand why you need to have a company arc, check the information at this site:

```
http://www.alvestrand.no/objectid/2.16.840.1.113730.1.13.html
```

The site contains information on Netscape-defined OID for an extension named Netscape Certificate Comment. Note that the OID assigned to this extension is hierarchical and it includes the Netscape company arc, which is `2.16.840.1.113730`. Every OID Netscape owns has this prefix.

When determining whether to add custom extension to certificates, keep in mind that if the extension exists in a certificate and if it is marked critical, the application validating the certificate must be able to interpret the extension (including the optional qualifiers, if any), or else it must reject the certificate. Since it's unlikely that all applications will be able to interpret your company's extensions (embedded in the form of OIDs), the PKIX standard recommends that the extension be always marked noncritical. For general guidelines on setting extensions in certificates, see "Certificate Extensions" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Overview of Extension-Specific Policy Modules

To enable you to add standard and private extensions to end-entity certificates, Certificate Management System provides a set of policy plug-in modules; each module enables you to add a particular extension to a certificate request. Plug-in modules are implemented as Java classes and are registered in the CMS policy framework. The Policy Plugin Registration tab of the CMS window (Figure 18.1) lists all the modules that are registered with a CMS instance.

When deciding whether to add any of the X.509 v3 certificate extensions, keep in mind that not all applications support X.509 v3 extensions. Among the applications that do support extensions, not all applications will recognize

every extension. For general guidelines on using extensions in certificates, see "Certificate Extensions" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 18.1 Extension policy modules registered with a Certificate Manager

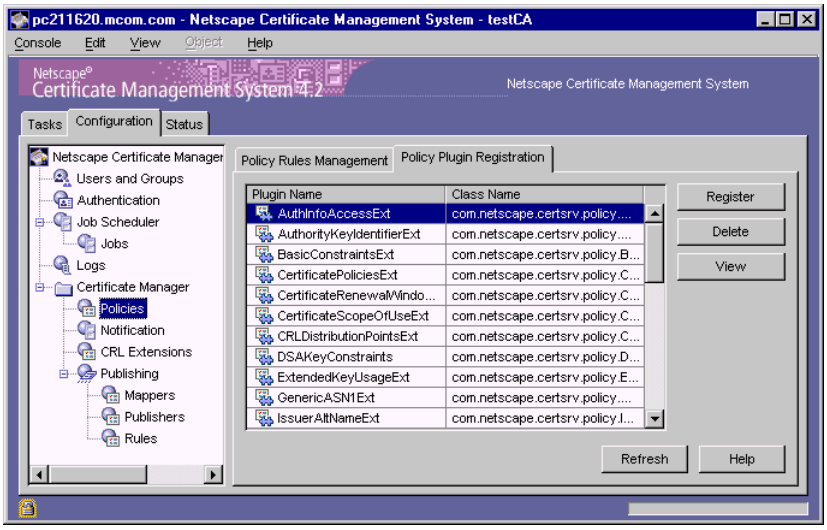


Table 18.1 lists extension-specific policy modules that are installed with a Certificate Manager. A Registration Manager installation also includes all the modules, expect for the ones noted below:

- `AuthorityKeyIdentifierExt`
- `BasicConstraintsExt`
- `NameConstraintsExt`
- `PolicyConstraintsExt`
- `PolicyMappingsExt`

You can use these modules to configure a Certificate Manager and Registration Manager to add extensions to certificates. Both subsystems add extensions to a certificate request when it undergoes policy processing (see "Policy Processor" on page 497). Keep in mind that the changes made to a request by a Registration Manager may be overwritten by a Certificate Manager when it subjects the request to its own policy checks.

Table 18.1 Default extension-specific policy plug-in modules

Plug-in module	Function
`AuthInfoAccessExt`	Adds the *Authority Information Access* extension to certificates. For details, see "Authority Information Access Extension Policy" on page 558.
`AuthorityKeyIdentifierExt`	Adds the *Authority Key Identifier* extension to certificates. For details, see "Authority Key Identifier Extension Policy" on page 566.
`BasicConstraintsExt`	Adds the *Basic Constraints* extension to certificates. For details, see "Basic Constraints Extension Policy" on page 570.
`CertificatePoliciesExt`	Adds the *Certificate Policies* extension to certificates. For details, see "Certificate Policies Extension Policy" on page 574.
`CertificateRenewalWindowExt`	Adds the *Certificate Renewal Window* extension to certificates. For details, see "Certificate Renewal Window Extension Policy" on page 580.
`CertificateScopeOfUseExt`	Adds the *Certificate Scope of Use* extension to certificates. For details, see "Certificate Scope of Use Extension Policy" on page 586.
`CRLDistributionPointsExt`	Adds the *CRL Distribution Points* extension to certificates. For details, see "CRL Distribution Points Extension Policy" on page 591.
`ExtendedKeyUsageExt`	Adds the *Extended Key Usage* extension to certificates. For details, see "Extended Key Usage Extension Policy" on page 598.
`GenericASN1Ext`	Adds ASN.1 type custom extension to certificates. For details, see "Generic ASN.1 Extension Policy" on page 605.
`IssuerAltNameExt`	Adds the *Issuer Alternative Name* extension to certificates. For details, see "Issuer Alternative Name Extension Policy" on page 612.
`KeyUsageExt`	Adds the *Key Usage* extension to certificates. For details, see "Key Usage Extension Policy" on page 618.
`NameConstraintsExt`	Adds the *Name Constraints* extension to certificates. For details, see "Name Constraints Extension Policy" on page 632.
`NSCCommentExt`	Adds the *Netscape Certificate Comment* extension to certificates. For details, see "Netscape Certificate Comment Extension Policy" on page 642.

Table 18.1 Default extension-specific policy plug-in modules (Continued)

Plug-in module	Function
`NSCertTypeExt`	Adds the *Netscape Certificate Type* extension to certificates. For details, see "Netscape Certificate Type Extension Policy" on page 647.
`OCSPNoCheckExt`	Adds the *OCSP No Check* extension to certificates. For details, see "OCSP No Check Extension Policy" on page 653.
`PolicyConstraintsExt`	Adds the *Policy Constraints* extension to certificates. For details, see "Policy Constraints Extension Policy" on page 657.
`PolicyMappingsExt`	Adds the *Policy Mappings* extension to certificates. For details, see "Policy Mappings Extension Policy" on page 661.
`PrivateKeyUsagePeriodExt`	Adds the *Private Key Usage Period* extension to certificates. For details, see "Private Key Usage Period Extension Policy" on page 666.
`SubjectAltNameExt`	Adds the *Subject Alternative Name* extension to certificates. For details, see "Subject Alternative Name Extension Policy" on page 668.
`SubjectDirectoryAttributesExt`	Adds a *Subject Directory Attributes* extension to certificates. For details, see "Subject Directory Attributes Extension Policy" on page 675.
`SubjectKeyIdentifierExt`	Adds the *Subject Key Identifier* extension to certificates. For details, see "Subject Key Identifier Extension Policy" on page 679.

As indicated in Table 18.1, Certificate Management System enables you to add almost all of the extensions defined in the PKIX standard RFC 2459 (`http://www.ietf.org/rfc/rfc2459.txt`). All modules have three features in common, enabling you to specify these:

- Whether to add the extension to certificates.
- The certificates to which the extension is to be added.
- Whether to mark the extension critical or noncritical.

Additionally, the server also provides a module for adding any custom, ASN.1 type extensions. If you determine that the default policy modules do not meet your requirements entirely, you can develop a custom module using CMS SDK. It is available in the form of Java Docs at this location:

`<server_root>/cms_sdk/sdkdocs`

For general guidelines on developing custom policy modules and adding them to the CMS policy framework, take a look at the samples installed at these locations:

`<server_root>/cms_sdk/samples/policy` and
`<server_root>/cms_sdk/samples/exttools`

For instructions to configure a Certificate Manager and Registration Manager to use one or more of the policy modules, see "Configuring a Subsystem's Policies" on page 685.

Authority Information Access Extension Policy

The `AuthInfoAccessExt` plug-in module implements the authority information access extension policy. This policy enables you to configure Certificate Management System to add the *Authority Information Access Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension specifies how the application validating the certificate can access information, such as on-line validation services and CA policy statements, about the CA that has issued the certificate in which the extension appears. Note that this extension should not be used to point directly to the CRL location maintained by the CA; the CRL Distribution Points extension explained in "CRL Distribution Points Extension Policy" on page 591 allows you to reference to CRL locations.

The PKIX standard recommends that you may include the authority information access extension in end-entity and CA certificates and that the extension be marked noncritical. For general guidelines on setting the authority information access extension, see "authorityInfoAccess" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The authority information access extension policy in Certificate Management System allows you to set the authority information access extension as defined in its X.509 definition. The policy enables you to specify any number of access points for CA information. For each access point, you can specify the access method, actual location that contains the additional information about the CA, and the mechanism for retrieving the information. The location can be specified in any of the following general-name forms: an rfc822name, a directory name, a DNS name, an EDI party name, a uniform resource indicator (URI), an IP address, an object identifier (OID), and any other name.

By default, the policy supports two access methods:

- `caIssuers` (this method is also identified by its OID, 1.3.6.1.5.5.7.48.2).

  As specified in the PKIX standard, you should use the `caIssuers` method when the additional information is a list of parent CAs or CAs that have issued certificates superior to the CA that issued the certificate containing the extension. The certificate-using application may use the list of parent CAs referenced by the extension to determine the certification path and to check whether the path terminates at a point trusted by the certificate user.

  When you use the `caIssuers` method, the access location referenced in the extension must take any of the following general-name forms:

  — Uniform resource identifier (URI) if the information is available via HTTP, FTP, or LDAP.

  — An X.500 directory name if the information is available via the directory access protocol (DAP).

  — An rfc822Name if the information is available via electronic mail.

- `ocsp` (this method is also identified by its OID, 1.3.6.1.5.5.7.48.1).

  The `ocsp` method indicates to the certificate-using client that it must use the OCSP protocol to access the location that contains additional information about the CA that has issued the certificate. You should use the `ocsp` method when you want to reference to the online validation authority that maintains the revocation status of certificates issued by the CA.

  When you use the `ocsp` method, the access location referenced in the extension must be a uniform resource indicator (URI); this means, the location type must be `URL` and the location value must be the complete URL (including the port number) at which the online validation authority for the CA is listening for OCSP requests from OCSP-compliant clients.

The built-in support for the `ocsp` access method and a URI value for the access location in the extension conform to the profile defined in RFC 2560 (see `http://www.ietf.org/rfc/rfc2560.txt`) for CAs that support the OCSP service. For details about OCSP support in Certificate Management System, see "Supported Methods for Verifying Revocation Status of Certificates" on page 724.

If you configure a Certificate Manager to publish CRLs to an OCSP responder and want to include the authority information access extension referencing to the responder, you should configure an instance of this policy as follows: access method is set to `ocsp`, name type is set to URI, and name value is set to the URL at which the OCSP responder listens to OCSP requests. This way, OCSP-compliant applications can verify the revocation status of certificates issued by the Certificate Manager by accessing the validation authority using the OCSP method.

Unlike some of the other policy modules, Certificate Management System does not create an instance of the `AuthInfoAccessExt` module during installation. If you want the server to add this extension to certificates, you must create an instance of the said module and configure it. For instructions, see "Step 4. Add New Policy Rules" on page 697).

AuthInfoAccessExt Module

The Java class that implements the authority information access extension policy is as follows:

```
com.netscape.certsrv.policy.AuthInfoAccessExt
```

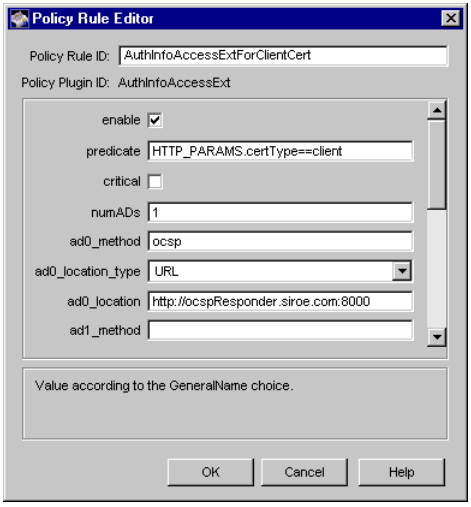
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.AuthInfoAccessExt.class=com.netscape.certsrv.policy.AuthInfoAccessExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `AuthInfoAccessExt`

Figure 18.2 shows how the configurable parameters for the `AuthInfoAccessExt` module are displayed in the CMS window.

Figure 18.2 Parameters defined in the AuthInfoAccessExt module



The configuration shown in Figure 18.2 creates a policy rule named `AuthInfoAccessExtForClientCert`, which enforces a rule that the server should add the authority information access extension to client certificates. The extension indicates that the online validation service (or the OSCSP responder) for the CA that has issued these certificates is at this URL:

```
http://ocspResponder.siroe.com:8000
```

The extension is marked noncritical (to comply with the PKIX recommendation).

Table 18.2 gives details about the configurable parameters defined in the `AuthInfoAccessExt` module.

Table 18.2 Description of parameters defined in the AuthInfoAccessExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the authority information access extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`numADs`	Specifies the total number of access locations to be contained or allowed in the extension. By default, this field is set to `3` and the UI shows fields for configuring three locations. You can change the total number of locations by changing the value assigned to this parameter; there's no restriction on the total number of locations you can include in the extension. Note that each location has its own set of configuration parameters and you must specify appropriate values for each of those parameters; otherwise the policy rule will return an error. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numADs` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no locations can be contained in the extension. • `n` specifies the total number of locations to be included in the extension; it must be an integer greater than zero. The default value is `3`. Example: `2`

Table 18.2 Description of parameters defined in the AuthInfoAccessExt module (Continued)

Parameter	Description
`ad<n>_method`	Specifies the access method for retrieving additional information about the CA that has issued the certificate in which the extension appears. Permissible values: • `ocsp` (or `1.3.6.1.5.5.7.48.1`). • `caIssuers` (or `1.3.6.1.5.5.7.48.2`). Example 1: `ocsp` Example 2: `1.3.6.1.5.5.7.48.1`
`ad<n>_location_type`	Specifies the general-name type for the location that contains additional information about the CA that has issued the certificate in which this extension appears. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URL`, `iPAddress`, `OID`, or `otherName`. • Select `rfc822Name` if the location is an Internet mail address. • Select `directoryName` if the location is an X.500 directory name. • Select `dNSName` if the location is a DNS name. • Select `ediPartyName` if the location is a EDI party name. • Select `URL` if the location is a uniform resource identifier (default). • Select `iPAddress` if the location is an IP address. • Select `OID` if the location is an object identifier. • Select `otherName` if the location is in any other name form. Example: `URL`
`ad<n>_location`	Specifies the address or location to get additional information about the CA that has issued the certificate in which this extension appears. Permissible values: Depends on the location type you specified in the `ad<n>_location_type` field. • If you selected `rfc822Name`, the value must be a valid Internet mail address in the `local-part@domain` format; see the definition of an `rfc822Name` as defined in RFC 822 (`http://www.ietf.org/rfc/rfc0822.txt`). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example: `ocspResponder@siroe.com`

Table 18.2 Description of parameters defined in the AuthInfoAccessExt module (Continued)

Parameter	Description
	• If you selected `directoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`). Note that RFC 2253 replaces RFC 1779. Example: `CN=corpDirectory, OU=IS, O=Siroe.com, C=US` • If you selected `dNSName`, the value must be a valid domain name in the preferred-name syntax as specified in RFC 1034 (`http://www.ietf.org/rfc/rfc1034.txt`). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as `rfc822Name`. Example: `ocspResponder.siroe.com` • If you selected `ediPartyName`, the value must be an IA5String. Example: `Siroe Corporation` • If you selected `URL`, the value must be a non-relative universal resource identifier (URI) following the URL syntax and encoding rules specified in RFC 1738 (`http://www.ietf.org/rfc/rfc1738.txt`). That is, the name must include both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. Example: `http://ocspResponder.siroe.com:8000` • If you selected `iPAddress`, the value must be a valid IP address specified in dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`.

Table 18.2 Description of parameters defined in the AuthInfoAccessExt module (Continued)

Parameter	Description
	• If you selected `OID`, the value must be a unique, valid OID specified in dot-separated numeric component notation. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.55.6.5.99` • If you selected `otherName`, the value must be the absolute path to the file containing the base-64 encoded string of the location. Example: `/usr/netscape/server4/ext/aia/othername.txt`

Authority Key Identifier Extension Policy

The `AuthorityKeyIdentifierExt` plug-in module implements the authority key identifier extension policy. This policy enables you to configure Certificate Management System to add the *Authority Key Identifier Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension is used to identify the public key that corresponds to the private key used by a CA to sign certificates.

You should consider adding this extension to all certificates, especially CA certificates, issued by Certificate Management System. The reason is, in certain situations, a CA's public key may change (for example, when the key gets updated) or the CA may have multiple signing keys (either due to multiple concurrent key pairs or due to key changeover). In these cases, the CA ends up with more than one distinct key. When verifying a signature on a certificate, other applications need to know which key was used in the signature. The extension, if present in a certificate, enables applications (those that can use the extension) to identify the correct key to use in situations when multiple keys exist; the extension specifies the public key to be used to verify the signature on the certificate.

For general guidelines on setting the authority key identifier extension, see "authorityKeyIdentifier" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide.*

The authority key identifier extension policy in Certificate Management System allows setting of the authority key identifier extension as defined in its X.509 definition with *key identifiers*. The policy enables you to specify what is to be done if the CA certificate does not have a subject key identifier extension—whether to use the a SHA-1 hash of the CA's subject public key information (carries the public key and identifies the algorithm with which the key is used) or skip adding the authority key identifier extension itself. For information on setting the subject key identifier extension in certificates, see "Subject Key Identifier Extension Policy" on page 679.

Note that PKIX and Federal PKI standards recommend against the use of `authorityCertIssuer` and `authorityCertSerialNumber` fields of the X.509 definition.

If enabled, the policy does the following:

- Sets the authority key identifier extension in certificates using the CA's key identifier in the CA's subject key identifier extension, if it exists. In the absence of a subject key identifier extension, the policy does either of the following (as specified by the configuration):

  — Uses the SHA-1 hash of the CA's subject public key information as the key identifier. This option is compatible with Netscape Communicator when the CA does not have a subject public key identifier extension.

  — Does not set the authority key identifier extension.

- Adds a authority key identifier extension to an enrollment request if the extension does not already exist. If the extension exists in the request, for example from a CRMF request, the policy replaces the extension. In case of manual enrollments, after an agent approves the enrollment request, the policy accepts any authority key identifier extension that is already there.

During installation, Certificate Management System automatically creates an instance of the authority key identifier extension policy. See "AuthorityKeyIdentifierExt Rule" on page 570.

AuthorityKeyIdentifierExt Module

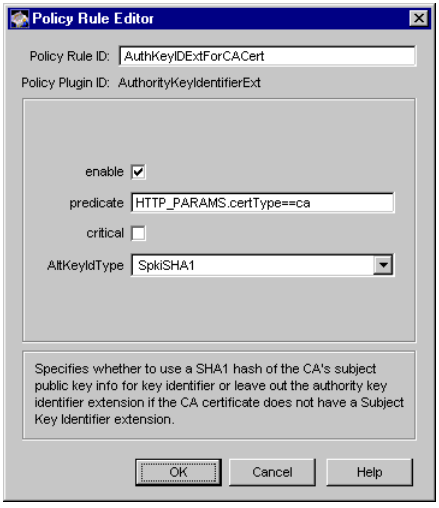
The Java class that implements the authority key identifier extension policy is as follows:

`com.netscape.certsrv.policy.AuthorityKeyIdentifierExt`

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.AuthorityKeyIdentifierExt.class=com.netscape.certsrv.policy.AuthorityKeyIdentifierExt`
- In the CMS window, the module is identified as follows: `AuthorityKeyIdentifierExt`

Figure 18.3 shows how the configurable parameters for the `AuthorityKeyIdentifierExt` module are displayed in the CMS window.

Figure 18.3 Parameters defined in the AuthorityKeyIdentifierExt module



The configuration shown in Figure 18.3 creates a policy rule named `AuthKeyIDExtForCACert`, which enforces a rule that the server should set the authority key identifier extension in all CA certificates.

Table 18.3 gives details about each of these parameters.

Table 18.3 Description of parameters defined in the AuthorityKeyIdentifierExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the authority key identifier extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`AltKeyIdType`	Specifies what should be done if the CA certificate does not have a Subject Key Identifier extension. Permissible values: `SpkiSHA1` or `None`. • Select `SpkiSHA1` if you want the server to use a SHA-1 hash of the CA's subject public key information (default). • Select `None` if you don't want the server to set the authority key identifier extension in certificates. Example: `SpkiSHA1`

AuthorityKeyIdentifierExt Rule

The rule named `AuthorityKeyIdentifierExt` is an instance of the `AuthorityKeyIdentifierExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the extension gets added to all certificates the server issues.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The rule specifies that a SHA-1 hash of the CA's subject public key info be used if the CA certificate does not have a Subject Key Identifier extension (`AltKeyIdType=SpkiSHA1`).

For details on individual parameters defined in the rule, see Table 18.3 on page 569. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Basic Constraints Extension Policy

The `BasicConstraintsExt` plug-in module implements the basic constraints extension policy. This policy enables you to configure Certificate Management System to add the *Basic Constraints Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in certificates. The extension identifies whether the Certificate Manager is a CA. In addition, the extension is also used during the certificate chain verification process to identify CA certificates and to apply certificate chain-path length constraints.

You should consider adding this extension to all CA certificates (root as well as subordinate) issued by Certificate Management System. The current PKIX standard requires that this extension be marked critical and that it appear in all CA certificates. The standard also recommends that the extension should not

appear in end-entity certificates. For general guidelines on setting the basic constraints extension, see "basicConstraints" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Because the basic constraints extension is a critical extension and is used by applications to determine the path length during certificate validation to chain up to the trusted CA, it's important that you set this extension correctly.

Also note that when a user submits a certificate request using the manual-enrollment method, the basic constraints extension is set on that request as per the configured policy, and then the request is queued for agent approval. When an agent approves the request, it is subjected to the configured policy again. If there's a change in the configuration of the basic constraints extension, the server may reject the agent-approved request. For the server to approve the request, the user will have to resubmit the request.

During installation, Certificate Management System automatically creates an instance of the basic constraints extension policy. See "BasicConstraintsExt Rule" on page 574.

BasicConstraintsExt Module

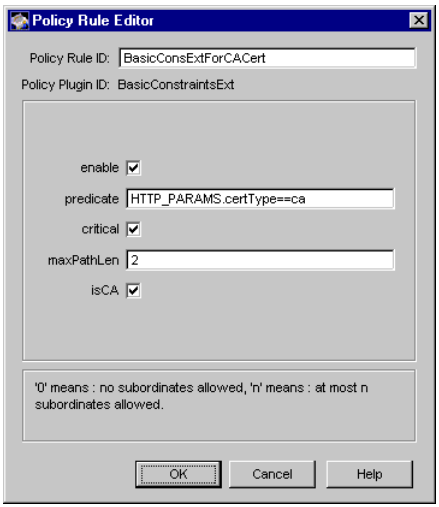
The Java class that implements the basic constraints extension policy is as follows:

```
com.netscape.certsrv.policy.BasicConstraintsExt
```

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.BasicConstraintsExt.class=com.netscape.certsrv.policy.BasicConstraintsExt`
- In the CMS window, the module is identified as follows: `BasicConstraintsExt`

Figure 18.4 shows how the configurable parameters for the `BasicConstraintsExt` module are displayed in the CMS window.

Figure 18.4 Parameters defined in the BasicConstraintsExt module



The configuration shown in Figure 18.4 creates a policy rule named `BasicConsExtForCACert`, which enforces a rule that the server should set the basic constraints extension in all CA certificates.

Table 18.4 gives details about each of these parameters.

Table 18.4 Description of parameters defined in the BasicConstraintsExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the basic constraints extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see “Using Predicates in Policy Rules” on page 490. Example: `HTTP_PARAMS.certType==ca`

Table 18.4 Description of parameters defined in the BasicConstraintsExt module (Continued)

Parameter	Description
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.
`maxPathLen`	Specifies the path length, the maximum number of CA certificates that may be chained below (subordinate to) the subordinate CA certificate being issued. Note that the path length you specify affects the number of CA certificates to be used during certificate validation. The chain starts with the end-entity certificate being validated and moving up the chain. The `maxPathLen` parameter has no effect if the extension is set in end-entity certificates. Permissible values: `0` or `n`. Make sure that the value you choose is less than the path length specified in the Basic Constraints extension of the CA signing certificate (owned by the CA that will issue these certificates). • `0` specifies that no subordinate CA certificates are allowed below the subordinate CA certificate being issued—that is, only an end-entity certificate may follow in the path. • `n` must be an integer greater than zero. It specifies at the most n subordinate CA certificates are allowed below the subordinate CA certificate being used. • If you leave the field blank, the path length defaults to a value that is determined by the path length set on the Basic Constraints extension in the issuer's certificate. If the issuer's path length is unlimited, the path length in the subordinate CA certificate will also be unlimited. If the issuer's path length is an integer greater than zero, the path length in the subordinate CA certificate will be set to a value that's one less than the issuer's path length; for example, if the issuer's path length is 4, the path length in the subordinate CA certificate will be set to 3. Example: `2`
`isCA`	Specifies whether the certificate subject is a CA. If you select the option, the server checks the `maxPathLen` parameter and sets the specified path length in the certificate. If you deselect the option, the server treats the certificate subject as a non-CA and ignores the value specified for the `maxPathLen` parameter.

BasicConstraintsExt Rule

The rule named `BasicConstraintsExt` is an instance of the `BasicConstraintsExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is set (`predicate=HTTP_PARAMS.certType==ca`) so that the extension gets added to CA certificates only.
- The extension is marked critical to comply with the PKIX recommendation.
- The path length field (`maxPathLen`) is left blank so that it defaults to a value that is determined by the path length set on the Basic Constraints extension in the issuer's certificate.

For details on individual parameters defined in the rule, see Table 18.4 on page 572. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Certificate Policies Extension Policy

The `CertificatePoliciesExt` plug-in module implements the certificate policies extension policy. This policy enables you to configure Certificate Management System to add the *Certificate Policies Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in certificates. The extension contains a sequence of one or more policy statements, each indicating the policy under which the certificate has been issued and identifying the purposes for which the certificate may be used. Presence of this extension in certificates enables an application with specific policy requirements to compare its list of policies to the ones contained in a certificate during its validation; typically, such applications will have a list of policies (which they will accept) and compare the policies in the certificate to their list as a part validating the certificate.

To promote interoperatability, the PKIX standard recommends that the policy statements or information terms should be included in certificates in the form of object identifiers (OIDs). For more information on OIDs, see "Object Identifier" on page 553. This means, in order for the server to add this extension to any certificate it issues, you need to compose policy statements you want to include in the extension, define OIDs for these policy statements, and configure the server with these OIDs.

When determining whether to add this extension to certificates, keep in mind that if the extension exists in a certificate and if it is marked critical, the application validating the certificate must be able to interpret the extension (including the optional qualifiers, if any), or else it must reject the certificate. For general guidelines on setting the certificate policies extension, see "certificatePolicies" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The certificate policies extension policy in Certificate Management System enables you to set the extension with the following information:

- The name of the your company or organization.
- The OID assigned to the policy statement you want to include in the certificate.
- A pointer (URI) to the published Certification Practice Statement (CPS).

  Any company deploying its own PKI should make a CPS available to anyone who may come across certificates issued by the CA deployed in the PKI. (The reason for this is people outside the company intranet may receive signed email messages from an employee.) To see an example of a CPS, check this site: `http://people.netscape.com/shadow/cps.html`
- A textual user notice (which the application validating the certificate can interpret and display).

During installation, Certificate Management System automatically creates an instance of the certificate policies extension policy. See "CertificatePoliciesExt Rule" on page 580.

CertificatePoliciesExt Module

The Java class that implements the certificate policies extension policy is as follows:

`com.netscape.certsrv.policy.CertificatePoliciesExt`

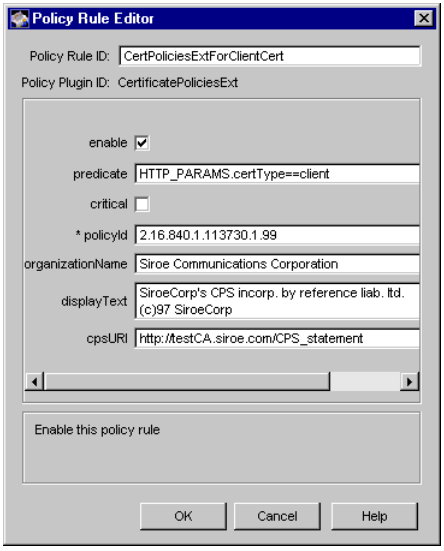
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.CertificatePoliciesExt.class=com.netscape.certsrv.policy.CertificatePoliciesExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `CertificatePoliciesExtPolicy`

Figure 18.5 shows how the configurable parameters for the `CertificatePoliciesExt` module are displayed in the CMS window.

Figure 18.5 Parameters defined in the CertificatePoliciesExt module



The configuration shown in Figure 18.5 creates a policy rule named `CertPoliciesExtForClientCert`, which enforces a rule that the server should set the certificate policies extension in all client certificates.

Table 18.5 gives details about each of these parameters.

Table 18.5 Description of parameters defined in the CertificatePoliciesExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. • Check the box to enable the rule (default). If you enable the rule and set the remaining parameters correctly, the server adds the certificate policies extension to certificates specified by the `predicate` parameter. • Uncheck the box to disable the rule. If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`policyId`	Specifies the OID assigned to the policy statement you want to include in the extension. If you specify a valid OID, the server includes the OID in the extension. The `policyId`, if specified, identifies by number a particular textual statement prepared by your organization (which is specified by the parameter named `organizationName`, listed next in this table). For example, it might identify the organization as Siroe Corporation and notice number `1.2.3.4.5.6.99`. Typically, applications validating the certificate will have a notice file containing the current set of notices for your company; these application will interpret the number in the certificate by extracting the notice text that corresponds to the number from the file and display it to the relying party. Permissible values: A unique, valid OID specified in dot-separated numeric component notation (see the example). Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `2.16.840.1.113730.1.99`

Table 18.5 Description of parameters defined in the CertificatePoliciesExt module (Continued)

Parameter	Description
`organizationName`	Specifies the name of the organization that owns the OID or is the owner of the policy statement referenced by the OID. Permissible values: The name of a company or its organizational unit. Example: `Siroe Corporation`
`displayText`	Specifies the textual statement to be included in certificates; this parameter corresponds to the `explicitText` field of the user notice. If you want to embed a textual statement (for example, your company's legal notice) in certificates, then add that statement here. The text you enter here will be displayed to a relying party when the certificate is used or viewed. Note that certain applications may not have the capability to display this text. Also, embedding a policy statement in a certificate increases its size. If you specify values for both `policyId` and `displayText` parameters and if the application software cannot locate the notice text indicated by the `policyId` parameter, then it is supposed to display the embedded notice; otherwise, it's supposed to display the information specified by the `policyId` parameter. (This feature is application specific and Certificate Management System has no control over it.) Permissible values: A string with up to 200 characters. Example: `SiroeCorp's CPS incorp. by reference liab. ltd. (c)97 SiroeCorp`
`cpsURI`	Specifies the location where the Certification Practice Statement published by the CA (that has issued the certificate) can be found. Permissible values: An IA5String value. The PKIX standard recommends that the pointer should be in the form of a URI. Example: `http://testCA.siroe.com/CPS_statement`

CertificatePoliciesExt Rule

The policy rule named `CertificatePoliciesExt` is an instance of the `CertificatePoliciesExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate field is left blank so that the extension gets added to all certificates.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- Other fields are left blank for you to enter the appropriate information.

For details on individual parameters defined in the rule, see Table 18.5 on page 577. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. Also, if you need to add additional instances, follow the instructions in "Step 4. Add New Policy Rules" on page 697. For example, if you want to include different policy statements in different types of certificates, you should create multiple instances of the policy module and configure each instance with the appropriate policy OID and predicate expression.

Certificate Renewal Window Extension Policy

The `CertificateRenewalWindowExt` plug-in module implements the certificate renewal window extension policy. This policy enables you to configure Certificate Management System to add the *Certificate Renewal Window Extension* to certificates. The extension, which must be noncritical, aids in managing the life cycle of a certificate by specifying a process to follow for renewing a certificate and by defining a time window when automatic renewal of the certificate should be attempted.

Every certificate issued by Certificate Management System has a validity period beyond which it cannot be used. In order to continue to participate in the PKI-using system beyond this validity period, the entity owning the certificate must

renew the certificate. Renewal of a certificate essentially means getting a new certificate for the existing key pair with a new validity time period (and updated attributes).

Once a certificate is issued, the owner of the certificate may attempt its renewal any time. To prevent certificate owners from renewing their certificates too often and thus reduce the overhead of processing new certificate requests, the CA can use a policy that restricts the time period when certificate renewal may occur. For example, the CA can use a policy that limits the renewal process to the last few weeks or days of validity of the certificate, thus defining a certificate renewal window. In general, the renewal window must be sufficient for the renewal to occur, but at the same time delay the renewal as long as possible to best utilize a certificate's life time.

The certificate-renewal process is often different than the enrollment process an entity uses to obtain the certificate; this is because the entity already owns a key pair that is associated with his or her identity. For example, in Certificate Management System, the certificate-renewal process for end users is different than the enrollment process they used to obtain the certificate. To renew their certificates, end users go to the certificate-renewal interface of Certificate Management System and submit their original certificates; for details, see "Authentication of End Users During Certificate Renewal" on page 312.

Because the renewal process requires end users to remember when their certificates expire and renew them before the expiry date, some clients provide built-in support for automated renewal. Inclusion of the certificate renewal window extension in certificates is useful in a PKI setup with such clients; such a setup eliminates the need for the owner of the certificate to manually submit a renewal request to the CA and install the renewed certificate. For example, assume you have deployed clients that can automatically submit certificate-renewal requests to Certificate Management System. If you issue certificates with the certificate renewal window extension to these clients, they can then read this extension for the renewal window and automatically get the certificate renewed from the CA during that window.

For a PKI setup without clients that can handle automated certificate renewals, Certificate Management System enables administrators to easily manage certificate renewals using the following features:

- The renewal notification job, which reminds users to renew their certificates before they expire.

- The renewal constraints policy, which determines whether expired certificates can be renewed; see "Renewal Constraints Policy" on page 518.
- The renewal validity constraints policy, which controls when users can renew their certificates and what should be the validity period in renewed certificates; see "Renewal Validity Constraints Policy" on page 525.

Unlike some of the other policy modules, Certificate Management System does not create an instance of the certificate renewal window extension policy during installation. If you want the server to add this extension to certificates, you must create an instance of the `CertificateScopeOfUseExt` module and configure it (see "Step 4. Add New Policy Rules" on page 697).

CertificateRenewalWindowExt Module

The Java class that implements the certificate renewal window extension policy is as follows:

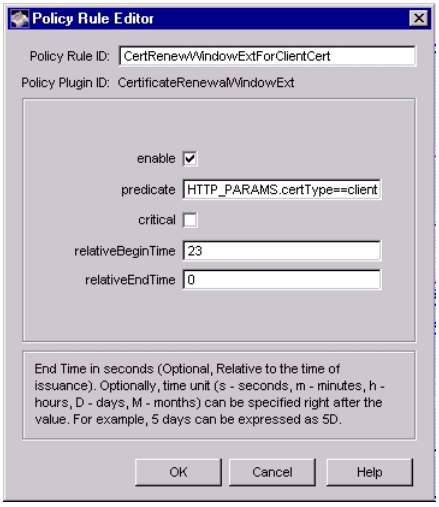
`com.netscape.certsrv.policy.CertificateRenewalWindowExt`

- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.CertificateRenewalWindowExt.class=`
  `com.netscape.certsrv.policy.CertificateRenewalWindowExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)
- In the CMS window, the module is identified as follows:
  `CertificateRenewalWindowExt`

Figure 18.6 shows how the configurable parameters for the `CertificateRenewalWindowExt` module are displayed in the CMS window.

Figure 18.6 Parameters defined in the CertificateRenewalWindowExt module



The configuration shown in Figure 18.6 creates a policy rule named `CertRenewWindowExtForClientCert`, which enforces a rule that the server should set the certificate renewal window extension in client certificates only; the renewal window starts 30 days before a certificate expires and ends with certificate expiration.

Table 18.6 gives details about each of these parameters.

Table 18.6 Description of parameters defined in the CertificateRenewalWindowExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. • Check the box to enable the rule (default). If you enable the rule and set the remaining parameters correctly, the server adds the certificate renewal window extension to certificates specified by the `predicate` parameter. • Uncheck the box to disable the rule. If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.6 Description of parameters defined in the CertificateRenewalWindowExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`relativeBeginTime`	Specifies the first time automatic renewal of certificate that contains the extension should be attempted. Permissible values: `0` or `n`. • `0` specifies that the renewal window begins at the same time the certificate is issued; the `beginTime` field of the extension will be set to the time of certificate issuance. • `n` specifies a future time for certificate renewal; the `beginTime` field of the extension will be set to the specified time since certificate issuance. You can specify the time period in seconds, minutes, hours, days, or months. Use the following suffixes to indicate the time unit. `s` - seconds `m` - minutes `h` - hours `D` - days `M` - months For example, if you're issuing certificates with a validity period of two years and want the renewal window to begin a month before the certificates expire, and want to specify the interval in months, you would enter `23M` in this field. To specify the same validity interval in seconds, you would set the value to `59616000s` (23 months x 30 days x 24 hours x 60 minutes x 60 seconds). Example: `23M`

Table 18.6 Description of parameters defined in the CertificateRenewalWindowExt module (Continued)

Parameter	Description
`relativeEndTime`	Specifies the last opportunity for automatic renewal of the certificate that contains this extension. Specifying a value for this parameter is optional; if you leave the field blank, the certificate-using application is expected to use the expiration date (`notAfter` value) in the certificate. Permissible values: `0` or `n`. • `0` specifies that the renewal window ends at the same time the certificate expires; the `endTime` field of the extension will be set to the time the certificate expires. • `n` specifies a past or future time, in seconds, by which the certificate must be renewed; the `endTime` field of the extension will be set to the specified time since certificate issuance. You can specify the time period in seconds, minutes, hours, days, or months. Use the following suffixes to indicate the time unit. `s` - seconds `m` - minutes `h` - hours `D` - days `M` - months For example, if you're issuing certificates with a validity period of two years and want the renewal window to end a month after the certificates expire, and want to specify the interval in months, you would enter `25M` in this field. On the other hand, if you want the renewal window to end 15 days before certificates expire, then you would set the value to `705D` ((23 months x 30 days) + 15 days). Note that if you choose to extend the renewal window after the expiration date of the certificate itself, your CA must maintain appropriate status information about the certificate during that window in order to allow appropriate authentication in the renewal process. (Automatic renewal may take place after the certificate has expired, when it is not valid for other purposes.) Example: `705D`

Certificate Scope of Use Extension Policy

The `CertificateScopeOfUseExt` plug-in module implements the certificate scope of use extension policy. This policy enables you to configure Certificate Management System to add the *Certificate Scope of Use Extension* to certificates. The extension enables you to specify a list of web sites that may request the use of a particular certificate for SSL client authentication, thus aiding certificate-using applications to select certificates to present to web sites and to control release of these certificates.

The SSL protocol provides a way for a client application to authenticate itself to a web site or server. SSL client authentication occurs upon request of the server, and proceeds by providing a certificate and a signature to the server. The client may have more than one certificate that could be used to perform this authentication. The SSL protocol provides a way for the server to indicate which certificate may be useful by listing issuing CAs in one of the SSL protocol messages.

By using a particular certificate for SSL client authentication, the client releases information about itself to the server. This information may include the name and key information contained in the certificate. It also releases the information that the client holds a certificate from a particular CA. This information may be of interest to the company running the server, for example to find users that have certificates from competing companies.

The certificate scope of use extension can be included in certificates to restrict the *scope-of-use* of the certificate for client authentication; the extension enables the certificate-using application to restrict the release of individual certificates to web sites requesting SSL client authentication.

The certificate scope of use extension policy in Certificate Management System enables you to include a list of name patterns that will match server DNS names where the certificate may be used. It's up to the certificate-using applications to use the values in this extension to filter the list of potential certificates to use for client authentication.

Unlike some of the other policy modules, Certificate Management System does not create an instance of the certificate scope of use extension policy during installation. If you want the server to add this extension to certificates, you must create an instance of the `CertificateScopeOfUseExt` module and configure it (see "Step 4. Add New Policy Rules" on page 697).

CertificateScopeOfUseExt Module

The Java class that implements the certificate scope of use extension policy is as follows:

`com.netscape.certsrv.policy.CertificateScopeOfUseExt`

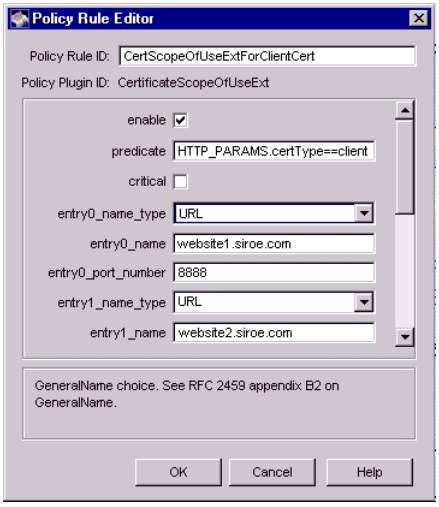
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.CertificateScopeOfUseExt.class=com.netscape.certsrv.policy.CertificateScopeOfUseExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `CertificateScopeOfUseExt`

Figure 18.7 shows how the configurable parameters for the `CertificateScopeOfUseExt` module are displayed in the CMS window.

Figure 18.7 Parameters defined in the CertificateScopeOfUseExt module



The configuration shown in Figure 18.7 creates a policy rule named `CertScopeOfUseExtForClientCert`, which enforces a rule that the server should set the certificate scope of use extension in client certificates only.

Table 18.7 gives details about each of these parameters.

Table 18.7 Description of parameters defined in the CertificateScopeOfUseExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. • Check the box to enable the rule (default). If you enable the rule and set the remaining parameters correctly, the server adds the certificate scope of use extension to certificates specified by the `predicate` parameter. • Uncheck the box to disable the rule. If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`numEntries`	Specifies the total number of sites to be contained or allowed in the extension. By default, this field is set to `3` and the UI shows fields for configuring three sites. You can change the total number of sites by changing the value assigned to this parameter; there's no restriction on the total number of sites you can include in the extension. Note that each site has its own set of configuration parameters and you must specify appropriate values for each of those parameters; otherwise the policy rule will return an error. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numEntries` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no sites can be contained in the extension. • `n` specifies the total number of sites to be included in the extension; it must be an integer greater than zero. The default is `3`. Example: `2`

Table 18.7 Description of parameters defined in the CertificateScopeOfUseExt module (Continued)

Parameter	Description
`entry<n>_name_type`	Specifies the general-name type for the site that you want to include in the extension. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URL`, `iPAddress`, `OID`, or `otherName`. • Select `rfc822Name` if the site is an Internet mail address. • Select `directoryName` if the site is an X.500 directory name. • Select `dNSName` if the site is a DNS name (default). • Select `ediPartyName` if the site is a EDI party name. • Select `URL` if the site is a uniform resource identifier. • Select `iPAddress` if the site is an IP address. • Select `OID` if the site is an object identifier. • Select `otherName` if the site is in any other name form. Example: `URL`
`entry<n>_name`	Specifies the general-name value for the site you want to include in the extension. Permissible values: Depends on the general-name type you selected in the `entry<n>_name_type` field. • If you selected `rfc822Name`, the value must be a valid Internet mail address in the `local-part@domain` format; see the definition of an `rfc822Name` as defined in RFC 822 (`http://www.ietf.org/rfc/rfc0822.txt`). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example: `webSite@siroe.com` • If you selected `directoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`). Note that RFC 2253 replaces RFC 1779. Example: `CN=corpDirectory, OU=IS, O=Siroe.com, C=US`

Table 18.7 Description of parameters defined in the CertificateScopeOfUseExt module (Continued)

Parameter	Description
	• If you selected `dNSName`, the value must be a valid domain name in the preferred-name syntax as specified in RFC 1034 (`http://www.ietf.org/rfc/rfc1034.txt`). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as `rfc822Name`. Example: `webSite.siroe.com` • If you selected `ediPartyName`, the value must be an IA5String. Example: `Siroe Corporation` • If you selected `URL`, the value must be a non-relative URI, including both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. Example: `http://webSite.siroe.com` • If you selected `iPAddress`, the value must be a valid IP address (IPv4 or IPv6) specified in dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`. • If you selected `OID`, the value must be a unique, valid OID specified in dot-separated numeric component notation. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.55.6.5.99`

Table 18.7 Description of parameters defined in the CertificateScopeOfUseExt module (Continued)

Parameter	Description
	• If you selected `otherName`, the value must be the absolute path to the file that contains the base-64 encoded string for the site. Example: `/usr/netscape/server4/ext/aia/othername.txt`
`entry<n>_port_number`	Specifies the port number. Example: 8888

CRL Distribution Points Extension Policy

The `CRLDistributionPointsExt` plug-in module implements the CRL distribution points extension policy. This policy enables you to configure Certificate Management System to add the *CRL Distribution Points Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. This extension, when present in a certificate, identifies one or more locations from where the application that is validating the certificate can obtain the CRL information (to verify the revocation status of the certificate).

For general guidelines on setting the CRL distribution points extension in certificates, see "cRLDistributionPoints" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The CRL distribution points extension policy in Certificate Management System enables you to specify pointers to one or more CRL locations. The pointers can be in these forms: the name of the X.500 directory that stores the CRL, the URI to the location that contains the CRL, or both.

Note that in the current implementation, the policy supports only two name forms for distribution points, X.500 Directory Name and URI; URIs described in this document support two CRL retrieval mechanisms, LDAP-based and HTTP-based. Optionally, each distribution point may contain a set of reason flags, indicating what revocation reasons are covered by the CRL at that location. Also, the distribution point location can be relative to the location of the issuer. In this last case, the `issuerName` and `issuerType` parameters should be included to give the location of the issuer.

You can modify the policy to support any name form by making appropriate changes to the sample code provided for this purpose. The sample code is located here:

```
<server_root>/cms_sdk/samples/policy/CRLDistributionPoints
```

During installation, Certificate Management System automatically creates an instance of the CRL distribution points extension policy. See "CRLDistributionPointsExt Rule" on page 597.

CRLDistributionPointsExt Module

The Java class that implements the CRL distribution points extension policy is as follows:

```
com.netscape.certsrv.policy.CRLDistributionPointsExt
```

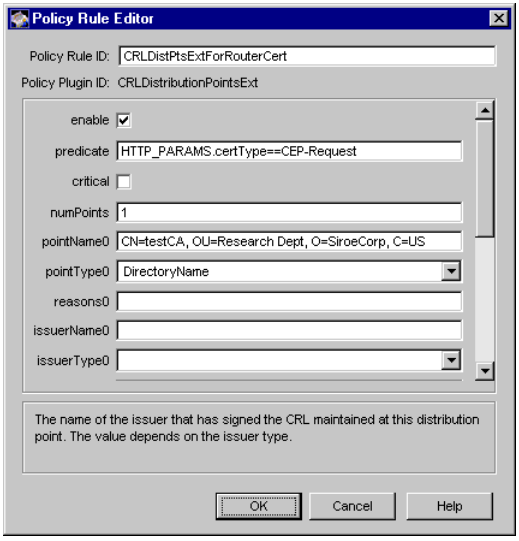
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.CRLDistributionPointsExt.class=com.netscape.certsrv.policy.CRLDistributionPointsExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `CRLDistributionPointsExt`

Figure 18.8 shows how the configurable parameters for the `CRLDistributionPointsExt` module are displayed in the CMS window.

Figure 18.8 Description of parameters defined in the CRLDistributionPointsExt module



The configuration shown in Figure 18.8 creates a policy rule named `CRLDistPtsExtForRouterCert`, which enforces a rule that the server should set the CRL distribution point extension in router certificates; the CRL location is a X.500 directory.

Table 18.8 gives details about each of these parameters.

Table 18.8 Description of parameters defined in the CRLDistributionPointsExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the CRL distribution points extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.8 Description of parameters defined in the CRLDistributionPointsExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==CEP-Request`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`numPoints`	Specifies the total number of CRL distribution points to be contained or allowed in the extension. By default, this field is set to `3` and the UI shows fields for configuring three distribution points. You can change the total number of distribution points by changing the value assigned to this parameter; there's no restriction on the total number of distribution points you can include in the extension. Note that each distribution point has its own set of configuration parameters and you must specify appropriate values for each of those parameters; otherwise the policy rule will return an error. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numPoints` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no distribution points can be contained in the extension. • `n` specifies the total number of distribution points to be included in the extension; it must be an integer greater than zero. The default is `3`. Example: `2`

Table 18.8 Description of parameters defined in the CRLDistributionPointsExt module (Continued)

Parameter	Description
`pointName<n>`	Specifies the name of the CRL distribution point. Permissible values: Any supported name forms. By default, the name can be in any of the following formats: • An X.500 directory name in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`); note that RFC 2253 replaces RFC 1779. For example, the name would look similar to the subject name in a certificate, like this: `CN=CA Central, OU=Research Dept, O=Siroe Corp, C=US` • A URI; for example, it would look similar to this: `http://testCA.siroe.com:80` • An `RDN` which specifies a location relative to the CRL Issuer. In this case, the value of the `pointType` attribute must be `RelativeToIssuer`.
`pointType<n>`	Specifies the type of the CRL distribution point. Permissible values: `DirectoryName`, `URI`, or `RelativeToIssuer`. The type you select must correspond to the value in the `pointName` field. • Select `DirectoryName` if the value in the `pointName` field is an X.500 directory name (default). • Select `URI` if the value in the `pointName` field is a uniform resource indicator. • Select `RelativeToIssuer` if the value in the `pointName` field is a location relative to the CRL Issuer. Example: `URI`
`reasons<n>`	Specifies revocation reasons covered by the CRL maintained at the distribution point. Permissible values: A comma-separated list of the following constants. • `unused` • `keyCompromise` • `cACompromise` • `affiliationChanged` • `superseded` • `cessationOfOperation` • `certificateHold` Example: `keyCompromise`

Table 18.8 Description of parameters defined in the CRLDistributionPointsExt module (Continued)

Parameter	Description
`issuerName<n>`	Specifies the name of the issuer that has signed the CRL maintained at distribution point. Permissible values: Any supported name forms. By default, the name can be in any of the following formats: • An X.500 directory name in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`); note that RFC 2253 replaces RFC 1779. For example, the name would look similar to this: `CN=CA Central, OU=Research Dept, O=Siroe Corp, C=US` • A URI; for example, it would look similar to this: `http://testCA.siroe.com:80`
`issuerType<n>`	Specifies the general-name type of the CRL issuer that has signed the CRL maintained at distribution point. Permissible values: `DirectoryName` or `URI`. The value you specify for this parameter must correspond to the value in the `issuerName` field. • Select `DirectoryName` if the value in the `issuerName` field is an X.500 directory name (default). • Select `URI` if the value in the `issuerName` field is a uniform resource indicator. Example: `DirectoryName`

CRLDistributionPointsExt Rule

The policy rule named `CRLDistributionPointsExt` is an instance of the `CRLDistributionPointsExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate field is left blank so that the extension gets added to all certificates.
- The extension is marked noncritical (to comply with the PKIX recommendation).

- Other fields are left blank for you to enter the appropriate information.

For details on individual parameters defined in the rule, see Table 18.8 on page 594. It is important that you review this rule and make the appropriate changes required by your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. Also, if you need to add additional instances, follow the instructions in "Step 4. Add New Policy Rules" on page 697. For example, if you want to include different CRL distribution points in different types of certificates, you should create multiple instances of the policy module and configure each instance with the appropriate CRL distribution point and predicate expression.

Extended Key Usage Extension Policy

The `ExtendedKeyUsageExt` plug-in module implements the extended key usage extension policy. This policy enables you to configure Certificate Management System to add the *Extended Key Usage Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension identifies one or more purposes—in addition to or in place of the basic purposes indicated in the key usage extension—for which the certified public key may be used. For example, if the key usage extension identifies a key to be used for signing, the extended key usage extension can further narrow down the usage of the key for signing OCSP responses only or for signing Java applets only. (For information on key usage extension, see "Key Usage Extension Policy" on page 618.)

The PKIX standard suggests that organizations can define their own extended key usage purposes, if there's a need. Each key purpose must be identified by an OID, which in turn must be defined in accordance with IANA or ITU-T Rec. X.660 | ISO/IEC/ITU 9834-1. The standard also recommends that the extension may be marked either critical or noncritical—mark the extension critical if you want to restrict the usage of the certificate only for one of the key-usage purposes indicated by the extension; mark the extension noncritical, when you want it to indicate the intended purposes of the key, and not restrict the use of the certificate to the indicated purposes (in this case, validating applications are expected to treat the extension as an advisory field and may use it to identify the key, not its usage purpose).

Table 18.9 lists the usages defined by PKIX for use with the extended key usage extension.

Table 18.9 PKIX usage definitions for the extended key usage extension

Usage	OID
Server authentication	1.3.6.1.5.5.7.3.1
Client authentication	1.3.6.1.5.5.7.3.2
Code signing	1.3.6.1.5.5.7.3.3
Email	1.3.6.1.5.5.7.3.4
IPSec end system	1.3.6.1.5.5.7.3.5
IPSec tunnel	1.3.6.1.5.5.7.3.6
IPSec user	1.3.6.1.5.5.7.3.7
Timestamping	1.3.6.1.5.5.7.3.8

For general guidelines on setting the extended key usage extension in certificates, see "extKeyUsage" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The extended key usage extension policy in Certificate Management System allows setting of the key usage extension as defined in its X.509 definition. The policy enables you to specify OIDs, that identify key usages, in the extension.

During installation, Certificate Management System automatically creates two instances of the extended key usage extension policy. See "CODESigningExt Rule" on page 602 and "OCSPSigningExt Rule" on page 603.

ExtendedKeyUsageExt Module

The Java class that implements the extended key usage extension policy is as follows:

`com.netscape.certsrv.policy.ExtendedKeyUsageExt`

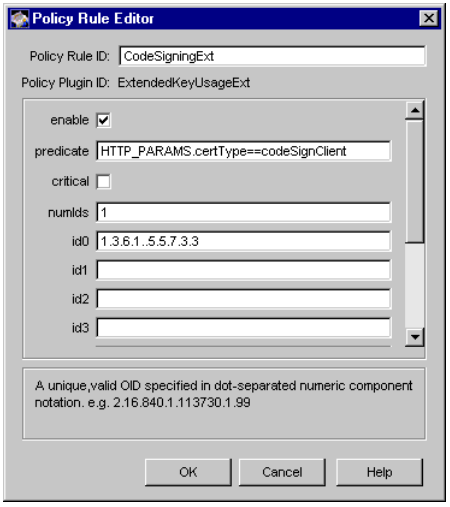
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.ExtendedKeyUsageExt.class=com.netscape.certsrv.policy.ExtendedKeyUsageExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `ExtendedKeyUsageExt`

Figure 18.9 shows how the configurable parameters for the `ExtendedKeyUsageExt` module are displayed in the CMS window.

Figure 18.9 Parameters defined in the ExtendedKeyUsageExt module



The configuration shown in Figure 18.9 creates a policy rule named `CodeSigningExt`, which enforces a rule that the extended key usage extension should be set in object-signing certificates.

Table 18.10 gives details about each of these parameters.

Table 18.10 Description of parameters defined in the ExtendedKeyUsageExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the extended key usage extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==codeSignClient`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.

Table 18.10 Description of parameters defined in the ExtendedKeyUsageExt module (Continued)

Parameter	Description
`numIds`	Specifies the total number of key-usage purposes to be contained or allowed in the extension. By default, this field is set to `10` and the UI shows fields for configuring ten key-usage purposes. You can change the total number by changing the value assigned to this parameter; there's no restriction on the total number of key-usage purposes you can include in the extension. Note that for each key-usage purpose, you must specify a valid OID; otherwise the policy rule will return an error. Configuration parameters for each key-usage purposes is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numIds` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no key-usage purposes can be contained in the extension. • `n` specifies the total number of key-usage purposes to be included in the extension; it must be an integer greater than zero. The default value is `10`. Example: `1`
`id<n>`	Specifies the OID that identifies a key-usage purpose. Permissible values: A unique, valid OID specified in the dot-separated numeric component notation. Depending on the key-usage purposes, you may choose to use the OIDs designated by PKIX (listed in Table 18.9 on page 598) or define your own OIDs. If you're defining your own OID, it should be in the registered subtree of IDs reserved for your company's use. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `2.16.840.1.113730.1.99`

CODESigningExt Rule

The rule named `CODESigningExt` is an instance of the `ExtendedKeyUsageExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is set (`HTTP_PARAMS.certType==codeSignClient`) so that the extension gets added to object signing certificates only—these certificates are used for signing objects.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The extension contains a single key-usage purpose, which is identified by an OID (`id0=1.3.6.1.5.5.7.3.3`). As shown in Table 18.9 on page 598, this OID is designated for code signing.

Note that this policy rule must remain enabled if you want Certificate Management System to issue object signing certificates with the correct extended key usage extension.

For details on individual parameters defined in the rule, see Table 18.10 on page 600. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

OCSPSigningExt Rule

The rule named `OCSPSigningExt` is an instance of the `ExtendedKeyUsageExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.

- The predicate expression is set (`HTTP_PARAMS.certType==ocspResponder`) so that the extension gets added to an OCSP responder certificate only—the certificate that corresponds to the key an online validation authority uses to sign OCSP responses.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The extension contains a single key-usage purpose, which is identified by an OID (`id0=1.3.6.1.5.5.7.3.9`).

Note that this policy rule must remain enabled if your PKI setup includes a CA-delegated OCSP responder and you want to issue an OCSP responder certificate to that server; the rule adds the extended key usage extension to an OCSP responder certificate indicating that the associated key can be used for signing OCSP responses.

Here's some background information that will help you understand why you should set this extension in OCSP responder certificates:

The online certificate status protocol (OCSP) enables OCSP-compliant applications to determine the revocation status of a certificate being validated. Certificate Management System supports the OCSP service—you can configure a Certificate Manager to publish CRLs to an online validation authority (also called OCSP responder); for details, see "Publishing of CRLs to an Online Validation Authority" on page 726. If you configure Certificate Management System to work with an OCSP responder, OCSP-compliant applications in your PKI setup will be able to do real-time verification of certificates by querying the OCSP responder for their revocation status. Note that these applications will be able to query the OCSP responder only if the certificate being validated includes the authority information access extension indicating the location of the OCSP responder; for information on adding this extension to certificates, see "Authority Information Access Extension Policy" on page 558.

When queried by an application on the status of a certificate, the OCSP responder sends a digitally signed response. To generate the signature, the responder needs to use a key. Because the signature needs to be verified by the application that sought the response, RFC 2560 recommends that the key used for signing an OCSP response must belong to one of the following:

- The CA that has issued the certificate, the revocation status of which is being requested.

- A trusted OCSP responder whose public key is trusted by the application that requested the revocation status of the certificate (as a part of validating the certificate).
- An OCSP responder that has been authorized by the CA (that has issued the certificate being validated) to sign OCSP responses for certificates issued by that CA.

  In this type of deployment, the CA authorizes a responder to sign OCSP responses on its behalf by issuing a specially marked certificate to the responder. This certificate is called the *OCSP responder certificate*, and it enables OCSP-compliant applications to identify the responder as a *CA-designated responder*—a responder authorized to sign OCSP responses for all certificates issued by the CA. The special marking that the CA includes in the certificate is the extended key usage extension with a unique value, `OCSPSigning`. This extension value indicates to OCSP-compliant applications that the key associated with the certificate can be used for signing OCSP responses.

If you want to deploy a CA-delegated OCSP responder, the `OCSPSigningExt` rule enables you to add the extended key usage extension (with `OCSPSigning` value) to the OCSP responder certificate. In addition to this extension, the responder's signing certificate should also include the *OCSP no check* extension. For details, see "OCSP No Check Extension Policy" on page 653.

Generic ASN.1 Extension Policy

The `GenericASN1Ext` plug-in module implements the generic ASN.1 extension policy. This policy enables you to configure Certificate Management System to add custom extensions to certificates. Using this policy, you can add as many ASN.1 type based-extensions as required without having to write any code. Further, it eliminates the dependency on the command-line tools for generating base-64 encoded standard extensions from the x.509 extension classes.

The generic extension policy in Certificate Management System accepts custom extensions in the form of object identifiers (OIDs) and values as DER-encoded extension values. That is, for the server to add a custom extension to certificates it issues, you need to first define the extension and then configure the server with extension details.

Similar to a standard extension, you define a custom extension by defining an OID and a ASN.1 structure.

- The OID must be specified in the dot-separated numeric component notation (for example, 2.5.29.35). Although you can invent your own OIDs for the purposes of evaluating and testing the server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs.

- The ASN.1 structure must be constructed from a sequence of DER-encoded extension values.

The resulting extension would look similar to the way a standard extension appears in certificates (as defined in RFC 2459):

```
Extension  ::=  SEQUENCE  {
   extnID      OBJECT IDENTIFIER,
   critical    BOOLEAN DEFAULT FALSE,
   extnValue   OCTET STRING  }
```

In the policy configuration, the `extnID` field is defined by the `oid` parameter, the `critical` field is defined by the `critical` parameter, and the `extnValue` field is defined by evaluating the expression in the `pattern` parameter, which in turn is defined by the `attribute` parameters. See Table 18.11 on page 608 for details on individual parameters.

Typically, the application receiving the certificate checks the extension ID to determine if it can recognize the ID. If it can, it uses the extension ID to determine the type of value used. When adding your custom extension to certificates, keep in mind that if the extension exists in a certificate and if it is marked critical, the application validating the certificate must be able to interpret the extension, or else it must reject the certificate. Since it's unlikely that all applications will be able to interpret your custom extensions, you should consider marking these extensions noncritical.

Note that each instance of the policy can be configured to add one custom extension only. To configure the server to add multiple custom extensions, create multiple instances of the module, each with a distinct name and appropriate configuration values. Also note that the policy allows you to encode simple (possibly nested) SEQUENCEs. There is no support for CHOICE, SET, or ASN.1 tagging.

During installation, Certificate Management System automatically creates an instance of the generic ASN.1 extension policy. See "GenericASN1Ext Rule" on page 612.

GenericASN1Ext Module

The Java class that implements the generic extension policy is as follows:

`com.netscape.certsrv.policy.GenericASN1Ext`

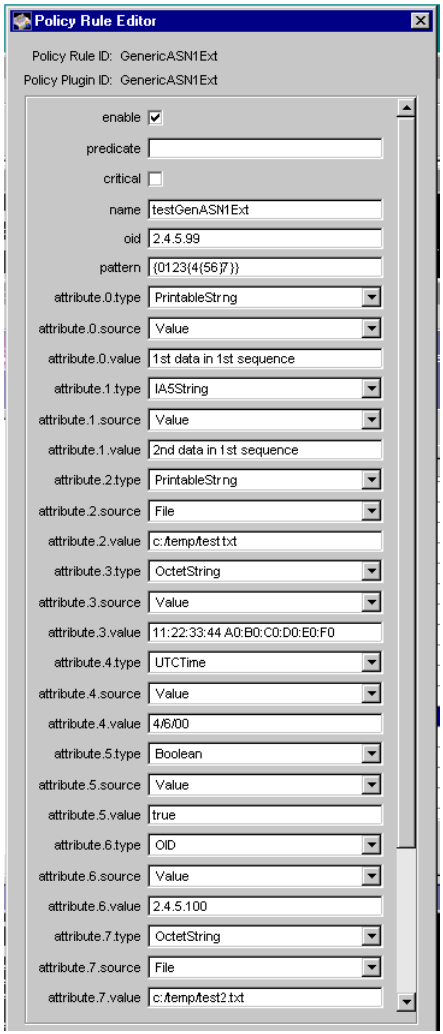
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.GenericASN1Ext.class=com.netscape.certsrv.policy.GenericASN1Ext`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `GenericASN1Ext`

Figure 18.10 shows how the configurable parameters for the `GenericASN1Ext` module are displayed in the CMS window.

Figure 18.10 Parameters defined in the GenericASN1Ext module



The configuration shown in Figure 18.10 defines a custom extension named `testGenASN1Ext` with OID 2.4.5.99. The extension is non-critical, and it will be added to all certificates issued by the server. The expected `dumpasn1` output (see "dumpasn1 Tool" on page 1195) of the resulting extension, would look like this:

```
337 30  148: . . . . SEQUENCE {
 340 06    3: . . . . . OBJECT IDENTIFIER '2 4 5 99'
 345 04  140: . . . . . OCTET STRING, encapsulates {
 348 30  137: . . . . . . . SEQUENCE {
 351 13   24: . . . . . . . . PrintableString '1st data in 1st sequence'
 377 16   24: . . . . . . . . IA5String '2nd data in 1st sequence'
 403 13   32: . . . . . . . . PrintableString 'This is 3rd data in 1st
                               sequence'
 437 04   10: . . . . . . . . OCTET STRING
               : . . . . . . . . . 11 22 33 44 A0 B0 C0 D0 E0 F0
 449 30   37: . . . . . . . . SEQUENCE {
 451 17   13: . . . . . . . . . UTCTime '000406070000Z'
 466 30    8: . . . . . . . . . SEQUENCE {
 468 01    1: . . . . . . . . . . BOOLEAN TRUE
 471 06    3: . . . . . . . . . . OBJECT IDENTIFIER '2 4 5 100'
               : . . . . . . . . . . }
 476 04   10: . . . . . . . . . OCTET STRING
               : . . . . . . . . . 11 22 33 44 A0 B0 C0 D0 E0 F0
               : . . . . . . . . . }
               : . . . . . . . . }
               : . . . . . . . }
               : . . . . . }
```

Table 18.11 describes the configurable parameters of the generic ASN.1 extension policy module.

Table 18.11 Description of parameters defined in the GenericASN1Ext module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the configured extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`

Table 18.11 Description of parameters defined in the GenericASN1Ext module (Continued)

Parameter	Description
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default). In general, you should make custom extensions noncritical if you want your certificates supported by other applications. (Other applications most likely will not understand your extension.)
`name`	Specifies the name of the extension. The name is displayed when users view the details of a certificate that includes the extension. Permissible values: A unique name that corresponds to the OID specified by the `oid` parameter. Example: `myCorp'sExtension`
`oid`	Specifies the OID assigned to the extension. Permissible values: A valid OID specified in dot-separated numeric component notation (see the example). Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.22.333.444.55.666`
`pattern`	Specifies the pattern of the extension. Permissible values: The pattern can be any sequence of supported ASN.1 type. Rules for formulating the pattern are as follows: • Each data component in the pattern must be represented by it's predefined attribute identifier, `0` to `9`, and each sequence must be grouped by a pair of curly brackets, `{}`. • Each attribute identifier represented in the pattern must be fully defined in the extension. For example, if you want to include attribute identifier 0, you must specify values for `attribure.0.type`, `attribure.0.source`, and `attribure.0.value` parameters. No default value is assigned to this parameter. Example: `{{012}34}`

Table 18.11 Description of parameters defined in the GenericASN1Ext module (Continued)

Parameter	Description
`attribute.<n>.type`	Specifies the data type for attribute `n`, where `n` is an identifier assigned to identify parameters pertaining to a specific attribute. The value of `n` can be `0` to `9`. Permissible values: `Integer`, `IA5String`, `OctetString`, `PrintableString`, `UTCtime`, `OID`, or `Boolean`. • Select `Integer` for extensions that have ASN.1 INTEGER values (default). It's case insensitive and accepts an integer in decimal notation as value. • Select `IA5String` for extensions that have ASN.1 IA5String values. It's case insensitive and accepts any normal string as value. • Select `OctetString` for extensions that have ASN.1 OCTET STRING values. It's case insensitive and the value is dependent on data source. If the data source is `Value`, the value must be in colon-separated, ASCII hexadecimal encoding notation. If the data source is `File`, the server reads the attribute value from the file specified. • Select `PrintableString` for extensions that have ASN.1 PrintableString values. It's case insensitive and accepts any normal string as value. • Select `UTCTime` for site-defined extensions that have ASN.1 UTCTime values. • Select `OID` for extensions that have ASN.1 OBJECT IDENTIFIER values. • Select `Boolean` for extensions that have ASN.1 BOOLEAN values. It's case insensitive and accepts `true` or `false` as value. Example: `Integer`
`attribute.<n>.source`	Specifies the data source for attribute `n` in the extension, where `n` is an identifier assigned to identify parameters pertaining to a specific attribute. The value of `n` can be `0` to `9`. In some cases, it may be preferable to put the value of an attribute in a file, instead of specifying it in the configuration parameters. This may be the case if the value of the attribute is a long text file or octet-string, for example. Permissible values: `Value` or `File`. (The attribute's `value` parameter is interpreted according to the value specified for this parameter.) • `Value` specifies that the attribute's `value` parameter is literally the value to be inserted in the extension (default). • `File` specifies that the attribute's `value` parameter is a fully-qualified pathname of a file containing the value to be inserted in the extension. Example: `Value`

Table 18.11 Description of parameters defined in the GenericASN1Ext module (Continued)

Parameter	Description
`attribute.<n>.value`	Specifies the data value for attribute `n`, where `n` is an identifier assigned to identify parameters pertaining to a specific attribute. The value of `n` can be `0` to `9`. Permissible values: Depends on the data type and source you selected. • If the data type is `Integer`, enter an integer in decimal notation as value. For example, `1234567890`. • If the data type is `IA5String`, enter a normal string as value. For example, `Test of IA5String`. • If the data type is `OctetString` and if the data source is `Value`, enter the value in colon-separated ASCII hexadecimal encoding notation; for example, `11:22:33:44:A0:B0:C0:D0:E0:F0`. If data source is `File`, enter the complete file path, including the filename, in the specified format. When specifying file path in a Window NT system do not use the NT native file separator, the backward slash (\). Use Unix style file separator, the forward slash (/), instead. For example, `C:/customExt/octet_string_value.txt`. • If the data type is `PrintableString`, enter a normal string as value. For example, `This_is_a_printable_string`. • If the data type is `UTCTime`, enter a date in `mm/dd/yy` format. For example, April 5, 2000 would be `4/5/00` and October 10, 2001 would be `10/10/01`. • If the data type is `OID`, enter a valid OID. For example, `11.33.234.99`. • If the data type is `Boolean`, enter `true` or `false` as value. For example, `true`.

GenericASN1Ext Rule

The rule named `GenericASN1Ext` is an instance of the `GenericASN1Ext` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate field is left blank so that the extension gets added to all certificates the server issues.
- The extension is marked noncritical (to comply with the PKIX recommendation).

- Other fields are left blank for you to enter the appropriate information.

For details on individual parameters defined in the rule, see Table 18.11 on page 608. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Issuer Alternative Name Extension Policy

The `IssuerAltNameExt` plug-in module implements the issuer alternative name extension policy. This policy enables you to configure Certificate Management System to add the *Issuer Alternative Name Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. This extension enables binding of or associating Internet style identities—such as Internet electronic mail address, a DNS name, an IP address, and a uniform resource indicator (URI)— with the certificate issuer.

For general guidelines on setting the issuer alternative name extension, see "issuerAltName" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide.*

The issuer alternative name extension policy in Certificate Management System allows setting of the issuer alternative name extension as defined in its X.509 definition. The policy enables you to associate the following alternative identities to a CA, by including them in the extension:

- An rfc822 name
- A directory name
- A DNS name
- An EDI party name
- A uniform resource indicator (URI)
- An IP address
- An object identifier (OID)

- Other Name

Unlike some of the other policy modules, Certificate Management System does not create an instance of the issuer alternative name extension policy during installation. If you want the server to add this extension to certificates, you must create an instance of the `IssuerAltNameExt` module and configure it (see "Step 4. Add New Policy Rules" on page 697).

IssuerAltNameExt Module

The Java class that implements the issuer alternative name extension policy is as follows:

```
com.netscape.certsrv.policy.IssuerAltNameExt
```

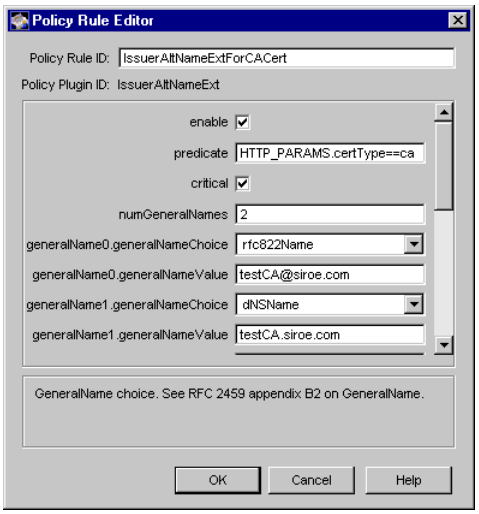
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.IssuerAltNameExt.class=com.netscape.certsrv.policy.IssuerAltNameExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `IssuerAltNameExt`

Figure 18.11 shows how the configurable parameters for the `IssuerAltNameExt` module are displayed in the CMS window.

Figure 18.11 Parameters defined in the IssuerAltNameExt module



The configuration shown in Figure 18.11 creates a policy rule named `IssuerAltNameExtForCACert`, which enforces a rule that the server should set the issuer alternative name extension in CA certificates only.

Table 18.12 gives details about each of these parameters.

Table 18.12 Description of parameters defined in the IssuerAltNameExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the issuer alternative name extension to all certificates specified by the `predicate` parameter. • If you disable the rule, the server doesn't add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`

Table 18.12 Description of parameters defined in the IssuerAltNameExt module (Continued)

Parameter	Description
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.
`numGeneralNames`	Specifies the total number of alternative names or identities permitted in the extension. Note that each name has a set of configuration parameters—`generalName<n>.generalNameChoice` and `generalName<n>.generalNameValue`—and you must specify appropriate values for each of those parameters; otherwise the policy rule will return an error. You can change the total number of identities by changing the value specified in this field; there's no restriction on the total number of identities you can include in the extension. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numGeneralNames` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no identities can be contained in the extension (default). • `n` specifies the total number of identities to be included in the extension; it must be an integer greater than zero. The default value is 8. Example: `2`

Table 18.12 Description of parameters defined in the IssuerAltNameExt module (Continued)

Parameter	Description
`generalName<n>.generalNameChoice`	Specifies the general-name type for the alternative name you want to include in the extension. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URL`, `iPAddress`, `OID`, or `otherName`. • Select `rfc822Name` if the alternative name is an Internet mail address (default). • Select `directoryName` if the alternative name is an X.500 directory name. • Select `dNSName` if the alternative name is a DNS name. • Select `ediPartyName` if the alternative name is a EDI party name. • Select `URL` if the alternative name is a uniform resource locator (URL). • Select `iPAddress` if the alternative name is an IP address. • Select `OID` if the alternative name is an object identifier. • Select `otherName` if the alternative name is in any other name form. Example: `rfc822Name`
`generalName<n>.generalNameValue`	Specifies the general-name value for the alternative name you want to include in the extension. Permissible values: Depends on the general-name type you selected in the `generalName<n>.generalNameChoice` field. • If you selected `rfc822Name`, the value must be a valid Internet mail address in the `local-part@domain` format; see the definition of an `rfc822Name` as defined in RFC 822 (`http://www.ietf.org/rfc/rfc0822.txt`). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example: `testCA@siroe.com` • If you selected `directoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`). Note that RFC 2253 replaces RFC 1779. Example, `CN=CA Corp,OU=Research Dept,O=Siroe Corp,C=US`

Table 18.12 Description of parameters defined in the IssuerAltNameExt module (Continued)

Parameter	Description
	• If you selected `dNSName`, the value must be a valid domain name in the preferred-name syntax as specified by RFC 1034 (`http://www.ietf.org/rfc/rfc1034.txt`). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as `rfc822Name`. Example: `testCA.siroe.com` • If you selected `ediPartyName`, the value must be an IA5String. Example: `Siroe Corporation` • If you selected `URL`, the value must be a non-relative universal resource identifier (URI) following the URL syntax and encoding rules specified in RFC 1738. That is, the name must include both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. Example, `http://testCA.siroe.com` • If you selected `iPAddress`, the value must be a valid IP address (IPv4 or IPv6) specified in dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`.

Table 18.12 Description of parameters defined in the IssuerAltNameExt module (Continued)

Parameter	Description
	• If you selected `OID`, the value must be a unique, valid OID specified in the dot-separated numeric component notation. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.55.6.5.99` • If you selected `otherName`, the value must be the absolute path to the file that contains the base-64 encoded string of the alternative name. Example: `/usr/netscape/server4/ext/ian/othername.txt`

Key Usage Extension Policy

The `KeyUsageExt` plug-in module implements the key usage extension policy. This policy enables you to configure Certificate Management System to add the *Key Usage Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension specifies the purposes for which the key contained in a certificate should be used—for example, it specifies whether the key should be used for data signing, key encipherment, or data encipherment—and thus enables you to restrict the usage of a key pair to predetermined purposes.

The key usage extension is a string of boolean bit-flags, each bit identifying the purpose for which a key is to be used. Table 18.13 lists the bits and their designated purposes.

Table 18.13 Key usage extension bits and designated purposes

Bit	Purpose
0	`digitalSignature`
1	`nonRepudiation`
2	`keyEncipherment`
3	`dataEncipherment`
4	`keyAgreement`

Table 18.13 Key usage extension bits and designated purposes (Continued)

Bit	Purpose
5	`keyCertSign`
6	`cRLSign`
7	`encipherOnly`
8	`decipherOnly`

You can restrict the purposes for which a key pair (and thus the corresponding certificate) should be used by setting the appropriate key-usage bits. For example, if you want to restrict a key pair to be used for digital signature only, when issuing the certificate you would add the key usage extension to the certificate with `digital_signature` bit (or bit 0) set. For general guidelines on setting the key usage extension in certificates, see "keyUsage" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide.*

Note that you can specify which bits in the extension are to be set on both server and client sides:

- On the server side, you set the bits by modifying the appropriate configuration parameters that are defined in the key usage extension policy.
- On the client side, bits set in the key usage extension are formed from pre-defined HTTP input variables that can be embedded as hidden values in the enrollment forms. You specify which bits are to be set by adding the appropriate HTTP variables to the enrollment forms. Table 18.14 lists the HTTP input variables that correspond to key usage extension bits.

Note For all certificates, the key-usage-bits set on the server side (which is governed by the policy) override the ones set on the client side.

Table 18.14 HTTP input variables for key usage extension bits

HTTP input variable	Key usage extension bit
`digital_signature`	digitalSignature (bit 0)
`non_repudiation`	nonRepudiation (bit 1)
`key_encipherment`	keyEncipherment (bit 2)
`data_encipherment`	dataEncipherment (bit3)

Table 18.14 HTTP input variables for key usage extension bits (Continued)

HTTP input variable	Key usage extension bit
`key_agreement`	keyAgreement (bit4)
`key_certsign`	keyCertsign (bit5)
`crl_sign`	cRLSign (bit6)
`encipher_only`	encipherOnly (bit7)
`decipher_only`	decipherOnly (bit8)

During installation, Certificate Management System automatically creates multiple instances of the key usage extension policy suitable for various types of certificates that you may want the server to issue. The default instances are named as follows:

- `CMCertKeyUsageExt`

  For details, see "CMCertKeyUsageExt Rule" on page 626.

- `RMCertKeyUsageExt`

  For details, see "RMCertKeyUsageExt Rule" on page 627.

- `ServerCertKeyUsageExt`

  For details, see "ServerCertKeyUsageExt Rule" on page 628.

- `ClientCertKeyUsageExt`

  For details, see "ClientCertKeyUsageExt Rule" on page 629.

- `ObjSignCertKeyUsageExt`

  For details, see "ObjSignCertKeyUsageExt Rule" on page 631.)

It is important that you review each policy instance and make the appropriate changes required by your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Additionally, as you'll notice in Figure 18.13 through Figure 18.17, the default enrollment forms provided for requesting various types of certificates (see "Forms for Certificate Enrollment" on page 915) include the appropriate HTTP input variables that correspond to the key-usage bits. By default only variables that correspond to key-usage bits that need to be set are included in the form.

Typically, you won't have to change the key-usage bit setting by editing the enrollment forms as you can do this easily by making the appropriate changes to the policy instance (bits set on the server side override the ones set on the client side). However, if you want to add new variables on the client side, you can do that too. Be sure to add the new variable in the following format:

```
<input type="HIDDEN" name="variable_name" value=true>
```

where, `variable_name` can be any of the HTTP input variables listed in Table 18.14.

The value of an HTTP input variable corresponding to a key-usage bit must be either `true` or `false`; any other value is considered equivalent to `false`. For example, a value `tree` would be interpreted as `false` by the server. Note that values `true` and `false` are case insensitive.

KeyUsageExt Module

The Java class that implements the key usage extension policy is as follows:

```
com.netscape.certsrv.policy.KeyUsageExt
```

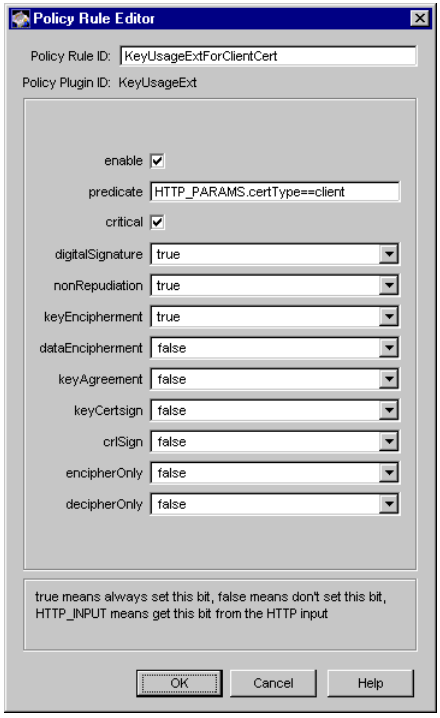
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.KeyUsageExt.class=com.netscape.certsrv.policy.KeyUsageExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `KeyUsageExt`

Figure 18.12 shows how the configurable parameters for the `KeyUsageExt` module are displayed in the CMS window.

Figure 18.12 Parameters defined in the KeyUsageExt module



The configuration shown in Figure 18.12 creates a policy rule named `KeyUsageExtForClientCert`, which enforces a rule that the server should set the key usage extension (`digitalSignature`, `nonRepudiation`, and `keyEncipherment` bits) in client certificates.

Table 18.15 gives details about each of these parameters.

Table 18.15 Description of parameters defined in the KeyUsageExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule, the server checks the key usage extension bits specified in the remaining fields, and adds the extension with those bits to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the key usage extension-specific bits specified in the policy configuration and in the enrollment forms.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.
`digitalSignature`	Specifies whether to set the `digitalSignature` bit (or bit 0) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `digitalSignature` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.

Table 18.15 Description of parameters defined in the KeyUsageExt module (Continued)

Parameter	Description
`nonRepudiation`	Specifies whether to set the `nonRepudiation` bit (or bit 1) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `nonRepudiation` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.
`keyEncipherment`	Specifies whether to set the `keyEncipherment` bit (or bit 2) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `keyEncipherment` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.
`dataEncipherment`	Specifies whether to set the `dataEncipherment` bit (or bit 3) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `dataEncipherment` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.

Table 18.15 Description of parameters defined in the KeyUsageExt module (Continued)

Parameter	Description
`keyAgreement`	Specifies whether to set the `keyAgreement` bit (or bit 4) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `keyAgreement` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.
`keyCertsign`	Specifies whether to set the `keyCertSign` bit (or bit 5) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `keyCertsign` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.
`cRLSign`	Specifies whether to set the `cRLSign` bit (or bit 6) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `CRLsign` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.

Table 18.15 Description of parameters defined in the KeyUsageExt module (Continued)

Parameter	Description
`encipherOnly`	Specifies whether to set the `encipherOnly` bit (or bit 7) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `encipherOnly` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.
`decipherOnly`	Specifies whether to set the `decipherOnly` bit (or bit 8) of the key usage extension in certificates specified by the `predicate` parameter. Permissible values: `true`, `false`, or `HTTP_INPUT`. • Select `true` if you want the server to set the bit (default). • Select `false` if you don't want the server to set the bit. • Select `HTTP_INPUT` if you want the server to check the certificate request for the HTTP input variable corresponding to the `decipherOnly` bit and set the bit accordingly. If the variable is set to `true`, the server sets the bit. If the variable doesn't exist or if it is set to `false` (or any other value), the server doesn't set the bit.

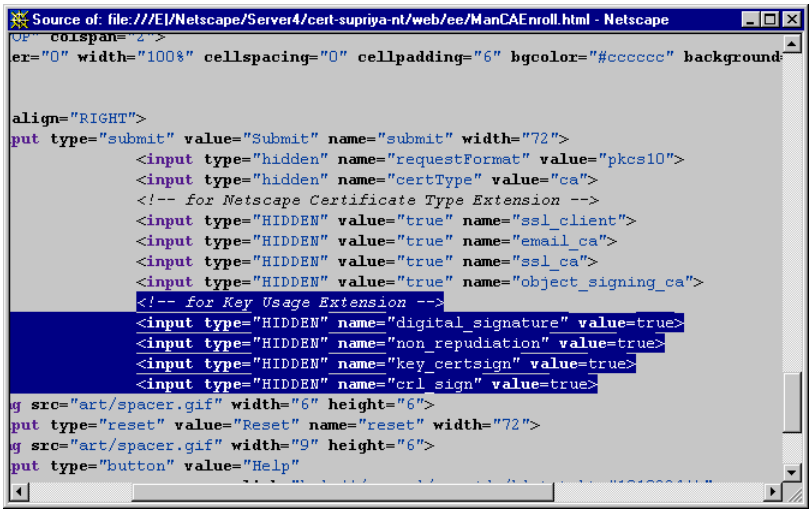
CMCertKeyUsageExt Rule

The policy rule named `CMCertKeyUsageExt` is an instance of the `KeyUsageExt` module. This rule is for setting the appropriate key-usage bits in Certificate Manager CA signing certificates; see "Certificate Manager's Key Pairs and Certificates" on page 226. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression (`predicate=HTTP_PARAMS.certType==ca`) ensures that the rule is applied only to CA signing certificate requests.

- The extension is marked noncritical (to comply with the PKIX recommendation).
- The server is configured to set `digitalSignature`, `nonRepudiation`, `keyCertsign`, and `cRLSign` bits in CA signing certificates. Notice that the key-usage bits specified in the default policy rule match the bits specified in the enrollment form (`ManCAEnroll.html`) for requesting CA signing certificates (see Figure 18.13).

Figure 18.13 Key usage bit-specific variables in the Certificate Manager enrollment form



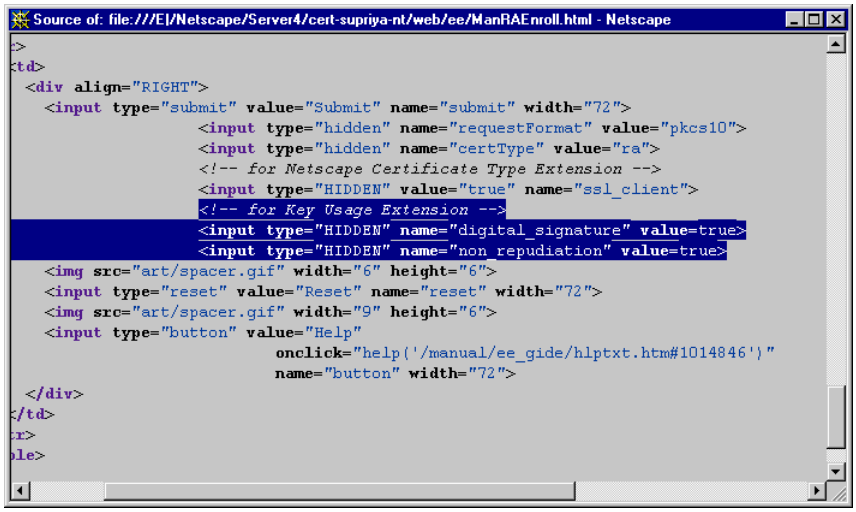
RMCertKeyUsageExt Rule

The policy rule named `RMCertKeyUsageExt` is an instance of the `KeyUsageExt` module. This rule is for setting the appropriate key-usage bits in Registration Managers' signing certificates; see "Registration Manager's Key Pairs and Certificates" on page 230. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression (`HTTP_PARAMS.certType==ra`) ensures that the rule is applied only to Registration Manager signing certificate requests.
- The extension is marked noncritical (to comply with the PKIX recommendation).

- The server is configured to set `digitalSignature` and `nonRepudiation` bits in Registration Manager signing certificates. Notice that the key-usage bits specified in the default policy rule match the bits specified in the enrollment form (`ManRAEnroll.html`) for requesting Registration Manager signing certificates (see Figure 18.14).

Figure 18.14 Key usage bit-specific variables in the Registration Manager enrollment form



ServerCertKeyUsageExt Rule

The policy rule named `ServerCertKeyUsageExt` is an instance of the `KeyUsageExt` module. This rule is for setting the appropriate key-usage bits in SSL server certificates. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression (`HTTP_PARAMS.certType==server`) ensures that the rule is applied only to SSL server certificate requests.
- The extension is marked noncritical (to comply with the PKIX recommendation).

- The server is configured to set `digitalSignature`, `nonRepudiation`, `keyEncipherment`, and `dataEncipherment` bits in SSL server certificates. Notice that the key-usage bits specified in the default policy rule match the bits specified in the enrollment form (`ManServerEnroll.html`) for requesting SSL server certificates (see Figure 18.15).

Figure 18.15 Key usage bit-specific variables in the SSL server certificate enrollment form

ClientCertKeyUsageExt Rule

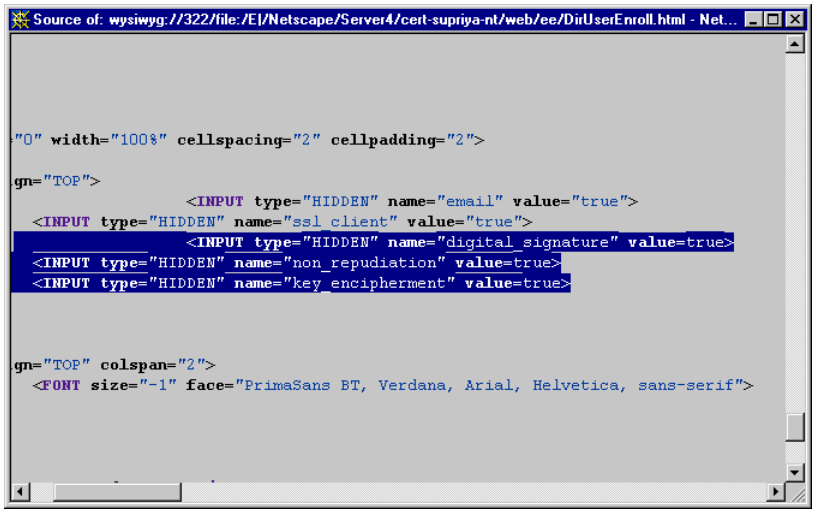
The policy rule named `ClientCertKeyUsageExt` is an instance of the `KeyUsageExt` module. This rule is for setting the appropriate key-usage bits in SSL client certificates. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression (`HTTP_PARAMS.certType==client`) ensures that the rule is applied only to SSL client certificate requests.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The server is configured to set `digitalSignature`, `nonRepudiation`, and `keyEncipherment` key-usage bits in SSL client certificates.

Notice that the key-usage bits specified in the default policy rule match the bits specified in the enrollment form for requesting SSL client certificates. Figure 18.16 shows the default directory-based enrollment form for end users with the information related to the key usage extension variables highlighted—it shows three of the total number of variables listed in Table 18.14 on page 619. Note that by default three key-usage bits—`digitalSignature`, `nonRepudiation`, and `keyEncipherment`—are enabled and the remaining bits are disabled.

Additionally, also notice the HTTP variables for the Netscape certificate type extension: the values indicate that the certificate is meant for S/MIME and SSL client authentication use only. (For details on Netscape certificate type extension, see “Netscape Certificate Type Extension Policy” on page 647.)

Figure 18.16 Key usage extension bits in the directory-based enrollment form



Keep in mind that for requesting client certificates, there are many enrollment forms. You may be using a combination of them:

- Certificate-based enrollment forms (`CertBasedDualEnroll.html`, `CertBasedEncryptionEnroll.html`, or `CertBasedSingleEnroll.html`)
- Directory-based enrollment form (`DirUserEnroll.html`)
- Directory- and PIN-based enrollment form (`DirPinUserEnroll.html`)
- Manual enrollment form (`ManUserEnroll.html`)

- NIS-based enrollment form (`NISEnroll.html`)
- Portal enrollment form (`PortalEnrollment.html`)

For details about these forms, see "Enrollment Forms" on page 361.

Each of these forms embed HTTP input variables (for key-usage bits) that are considered appropriate for the certificate being requested using that form. If you want, you may create additional instances of the key usage extension policy, one each for each client certificate enrollment form and configure these instances as appropriate. Be sure to use the correct predicate expression to distinguish the certificates to thus avoid setting incorrect bits.

ObjSignCertKeyUsageExt Rule

The policy rule named `ObjSignCertKeyUsageExt` is an instance of the `KeyUsageExt` module. This rule is for setting the appropriate key-usage bits in object signing certificates. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression (`predicate=HTTP_PARAMS.certType==objSignClient`) ensures that the rule is applied to only object signing certificate requests.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The server is configured to set `digitalSignature` and `keyCertsign` bits in object-signing certificates. Notice that the key-usage bits specified in the default policy rule match the bits specified in the enrollment form (`ManObjSignEnroll.html`) for requesting object-signing certificates (see Figure 18.17).

Figure 18.17 Key usage extension bits in the object signing certificate enrollment form

Name Constraints Extension Policy

The `NameConstraintsExt` plug-in module implements the name constraints extension policy. This policy enables you to configure Certificate Management System to add the *Name Constraints Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension is used in CA certificates to indicate a name space within which subject names or subject alternative names in subsequent certificates in a certification path or chain should be located.

Various standards describe how the name constraints extension should be processed during certificate verification. It's beyond the scope of this document to explain this. For general guidelines on setting the name constraints extension in certificates, see "nameConstraints" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The policy implemented in Certificate Management System allows setting of the name constraints extension in any form as defined in its X.509 definition; the policy enables you to specify the number of subtrees permitted and excluded in the extension. It is up to applications to process the extension as described in the standards.

During installation, Certificate Management System automatically creates an instance of the name constraints extension policy. See “NameConstraintsExt Rule” on page 641.

NameConstraintsExt Module

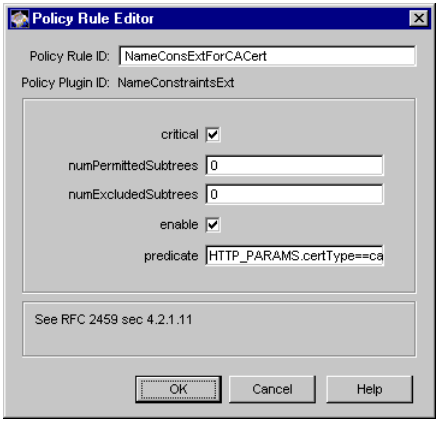
The Java class that implements the name constraints extension policy is as follows:

`com.netscape.certsrv.policy.NameConstraintsExt`

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.NameConstraintsExt.class=com.netscape.certsrv.policy.NameConstraintsExt`
- In the CMS window, the module is identified as follows: `NameConstraintsExt`

Figure 18.18 shows how the configurable parameters for the `NameConstraintsExt` module are displayed in the CMS window.

Figure 18.18 Parameters defined in the NameConstraintsExt module



The configuration shown in Figure 18.18 creates a policy rule named `NameConsExtForCACert`, which enforces a rule that the server should set the name constraints extension as a critical extension in CA certificates.

Table 18.16 gives details about each of these parameters.

Table 18.16 Description of parameters defined in the NameConstraintsExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the name constraints extension to all certificates specified by the `predicate` parameter. • If you disable the rule, the server doesn't add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.
`numPermittedSubtrees`	Specifies the total number of subtrees to be permitted in the extension. Note that each permitted subtree has a set of configuration parameters and you must specify appropriate values for each of these parameters; otherwise the policy rule will return an error. You can change the total number of permitted subtrees by changing the value in this field; there's no restriction on the total number of permitted subtrees you can include in the extension. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numPermittedSubtrees` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no permitted subtrees can be contained in the extension. • `n` specifies the total number of permitted subtrees to be included in the extension; it must be an integer greater than zero. The default value is 8. Example: `2`

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
`numExcludedSubtrees`	Specifies the total number of subtrees to be excluded in the extension. Note that each excluded subtree has a set of configuration parameters and you must specify appropriate values for each of these parameters; otherwise the policy rule will return an error. You can change the total number of excluded subtrees by changing the value in this field; there's no restriction on the total number of excluded subtrees you can include in the extension. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numExcludedSubtrees` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no excluded subtrees can be contained in the extension. • `n` specifies the total number of excluded subtrees to be included in the extension; it must be an integer greater than zero. The default value is 8. Example: `2`
`permittedSubtrees<n>.base.generalNameChoice`	Specifies the general-name type for the permitted subtree you want to include in the extension. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URI`, `iPAddress`, `registeredID`, or `otherName`. • Select `rfc822Name` if the subtree is an Internet mail address (default). • Select `directoryName` if the subtree is an X.500 directory name. • Select `dNSName` if the subtree is a DNS name. • Select `ediPartyName` if the subtree is a EDI party name. • Select `URL` if the subtree is a uniform resource locator. • Select `iPAddress` if the subtree is an IP address. • Select `OID` if the subtree is an object identifier. • Select `otherName` if the subtree is in any other name form. Example: `directoryName`

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
`permittedSubtrees<n>.base.generalNameValue`	Specifies the general-name value for the permitted subtree you want to include in the extension. Permissible values: Depends on the general-name type you selected in the `permittedSubtrees<n>.base.generalNameChoice` field. • If you selected `rfc822Name`, the value must be a valid Internet mail address in the `local-part@domain` format; see the definition of an `rfc822Name` as defined in RFC 822 (`http://www.ietf.org/rfc/rfc0822.txt`). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example: `testCA@siroe.com` • If you selected `directoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`). Note that RFC 2253 replaces RFC 1779. Example: `CN=SubCA, OU=Research Dept, O=SiroeCorp, C=US` • If you selected `dNSName`, the value must be a valid domain name in the preferred-name syntax as specified by RFC 1034 (`http://www.ietf.org/rfc/rfc1034.txt`). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as `rfc822Name`. Example: `testCA.siroe.com` • If you selected `ediPartyName`, the value must be a IA5String. Example: `Siroe Corporation` • If you selected `URL`, the value must be a non-relative universal resource identifier (URI) following the URL syntax and encoding rules specified in RFC 1738. That is, the name must include both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. Example: `http://testCA.siroe.com`

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
	• If you selected `iPAddress`, the value must be a valid IP address (IPv4 or IPv6) specified in the dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`. • If you selected `OID`, the value must be a unique, valid OID specified in dot-separated numeric component notation. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.55.6.5.99` • If you selected `otherName`, the value must be the absolute path to the file that contains the base-64 encoded string of the subtree. Example: `/usr/netscape/server4/ext/nc/othername.txt`
`permittedSubtrees<n>.min`	Specifies the minimum number of permitted subtrees. Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension. • `0` specifies that the minimum number of subtrees is zero (default). • `n` must be an integer that is greater than zero. It specifies at the most `n` subtrees are allowed. Example: `0`

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
`permittedSubtrees<n>.max`	Specifies the maximum number of permitted subtrees. Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension (default). • `0` specifies that the maximum number of subtrees is zero. • `n` must be an integer that is greater than zero. It specifies at the most `n` subtrees are allowed. Example: `1`
`excludedSubtrees<n>.base.generalNameChoice`	Specifies the general-name type for the excluded subtree you want to include in the extension. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URL`, `iPAddress`, `OID`, or `otherName`. • Select `rfc822Name` if the subtree is an Internet mail address. • Select `directoryName` if the subtree is an X.500 directory name. • Select `dNSName` if the subtree is a DNS name. • Select `ediPartyName` if the subtree is a EDI party name. • Select `URL` if the subtree is a uniform resource locator. • Select `iPAddress` if the subtree is an IP address. • Select `OID` if the subtree is an object identifier. • Select `otherName` if the subtree is in any other name form. Example: `OID`
`excludedSubtrees<n>.base.generalNameValue`	Specifies the general-name value for the excluded subtree you want to include in the extension. Permissible values: Depends on the general-name type you selected in the `excludedSubtrees<n>.base.generalNameChoice` field. • If you selected `rfc822Name`, the value must be a valid Internet mail address in the `local-part@domain` format; see the definition of an `rfc822Name` as defined in RFC 822 (`http://www.ietf.org/rfc/rfc0822.txt`). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example, `testCA@siroe.com`

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
	• If you selected `directoryName`, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see `http://www.ietf.org/rfc/rfc2253.txt`). Note that RFC 2253 replaces RFC 1779. Example: `CN=SubCA,OU=Research Dept,O=Siroe Corp,C=US` • If you selected `dNSName`, the value must be a valid domain name in the preferred-name syntax as specified by RFC 1034 (`http://www.ietf.org/rfc/rfc1034.txt`). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as `rfc822Name`. Example: `testCA.siroe.com` • If you selected `ediPartyName`, the value must be an IA5String. Example, `Siroe Corporation` • If you selected `URL`, the value must be a non-relative universal resource identifier (URI) following the URL syntax and encoding rules specified in RFC 1738. That is, the name must include both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. Example, `http://testCA.siroe.com` • If you selected `iPAddress`, the value must be a valid IP address (IPv4 or IPv6) specified in the dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`.

Table 18.16 Description of parameters defined in the NameConstraintsExt module (Continued)

Parameter	Description
	• If you selected `OID`, the value must be a unique, valid OID specified in dot-separated numeric component notation. Example: `1.2.3.4.55.6.5.99` • If you selected `otherName`, the value must be the absolute path to the file that contains the base-64 encoded string of the subtree. Example: `/usr/netscape/server4/ext/nc/othername.txt`
`excludedSubtrees<n>.min`	Specifies the minimum number of excluded subtrees. Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension. • `0` specifies that the minimum number of subtrees is zero (default). • `n` must be an integer that is greater than zero. It specifies at the most `n` subtrees are allowed. Example: `0`
`excludedSubtrees<n>.max`	Specifies the maximum number of excluded subtrees. Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension (default). • `0` specifies that the maximum number of subtrees is zero. • `n` must be an integer that is greater than zero. It specifies at the most `n` subtrees are allowed. Example: `1`

NameConstraintsExt Rule

The policy rule named `NameConstraintsExt` is an instance of the `NameConstraintsExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate expression is set (`predicate=HTTP_PARAMS.certType==ca`) so that the extension gets added to CA certificates only.
- The extension is marked critical (to comply with the PKIX recommendation).
- The total number of permitted subtrees to be contained in the extension is set to 3 (`numPermittedSubtrees=3`).
- The total number of excluded subtrees to be contained in the extension is set to 3 (`numExcludedSubtrees=3`).
- The maximum number of permitted subtrees is set to `-1` (`permittedSubtrees<n>.max=-1`) and the minimum number of permitted subtrees is set to `0` (`permittedSubtrees<n>.min=0`).
- The maximum number of excluded subtrees is set to `-1` (`excludedSubtrees<n>.max=-1`) and the minimum number of excluded subtrees is set to `0` (`excludedSubtrees<n>.min=0`).
- The general name type and value fields for each subtree are left blank for you to select or enter the appropriate values.

For details on individual parameters defined in the rule, see Table 18.16 on page 634. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Netscape Certificate Comment Extension Policy

The `NSCCommentExt` plug-in module implements the Netscape certificate comment extension policy. This policy enables you to configure Certificate Management System to add the *Netscape Certificate Comment Extension* (see `http://www.netscape.com/eng/security/cert-exts.html`) to certificates. The extension can be used to include textual comments in certificates. Applications that are capable of interpreting the comment may display it to a relying party when the certificate is used or viewed.

For general guidelines on setting the Netscape certificate comment extension, see "netscape-comment" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The Netscape certificate comment extension policy in Certificate Management System allows you to specify a textual statement or a comment to be included in certificates. You may choose to directly embed the text in the certificate itself or point to the file that contains the statement.

During installation, Certificate Management System automatically creates an instance of the Netscape certificate comment extension policy. See "NSCCommentExt Rule" on page 646.

NSCCommentExt Module

The Java class that implements the Netscape certificate comment extension policy is as follows:

`com.netscape.certsrv.policy.NSCCommentExt`

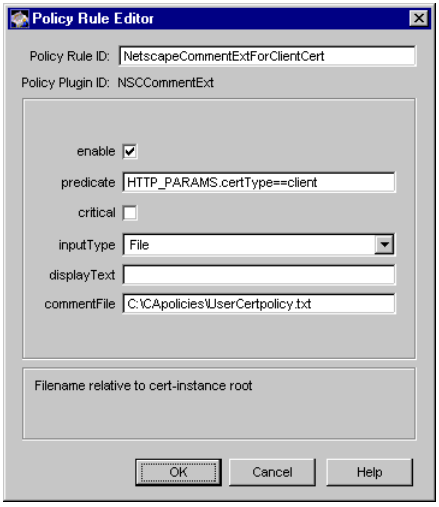
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.NSCCommentExt.class=com.netscape.certsrv.policy.NSCCommentExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `NSCCommentExt`

Figure 18.19 shows how the configurable parameters for the `NSCCommentExt` module are displayed in the CMS window.

Figure 18.19 Parameters defined in the NSCCommentExt module



The configuration shown in Figure 18.19 creates a policy rule named `NetscapeCommentExtForClientCert`, which enforces a rule that the server should set the Netscape certificate comment extension in client certificates.

Table 18.17 provides details for each of these parameters.

Table 18.17 Description of parameters defined in the NSCCommentExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the Netscape certificate comment extension to certificates specified by the `predicate` parameter. If you enable the policy without specifying values in `displayText` and `commentfile` fields, the server puts an empty string in the comment extension. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.17 Description of parameters defined in the NSCCommentExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`inputType`	Specifies whether to embed a textual statement or to include a pointer to file that contains the textual statement in certificates. The extension value is interpreted according to the value specified for this parameter. You should consider putting the textual statement in a file because certain applications may not have the capability to display the text embedded in certificates. Also, embedding a textual statement in a certificate increases its size. If you're using smart cards for generating and storing certificates, you may not want to embed textual statements in certificates because on a smart card the memory for a certificate may be limited. Permissible values: `Text` or `File`. • `Text` specifies that the textual statement—the value of the `displayText` field— should be inserted in the extension (default). • `File` specifies that the path to the file that contains the textual statement—the value of the `commentfile` field—should be inserted in the extension. Example: `File`
`displayText`	Specifies the textual statement that should be included in certificates. If you want to embed a textual statement (for example, your company's legal notice) in certificates, then add that statement here. The text you enter here will be displayed to a relying party when the certificate is used or viewed. Permissible values: A string with up to 200 characters. Example: `SiroeCorp's CPS incorp. by reference liab. ltd. (c)99 SiroeCorp`

Table 18.17 Description of parameters defined in the NSCCommentExt module (Continued)

Parameter	Description
`commentfile`	Specifies the path to the file that contains the textual statement that should be included in certificates; be sure to include the complete path, including the filename. Note that the existence of the file is not checked at the time of policy configuration. The filename will be checked when the policy is applied to a request. Example: `C:\CApolicies\UserCertpolicy.txt`

NSCCommentExt Rule

The policy rule named `NSCCommentExt` is an instance of the `NSCCommentExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The `predicate` field is left blank so that the extension gets added to all certificates.
- The extension is marked noncritical.
- The textual statement is to be embedded in certificates (`inputType=Text`).
- The `displayText` and `commentfile` fields are left blank for you to enter the appropriate information.

For details on individual parameters defined in the rule, see Table 18.17 on page 644. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Netscape Certificate Type Extension Policy

The `NSCertTypeExt` plug-in module implements the Netscape certificate type extension policy. This policy enables you to configure Certificate Management System to add the *Netscape Certificate Type* extension to certificates. The extension identifies the certificate type—for example, it identifies whether the certificate is a CA certificate, server SSL certificate, client SSL certificate, object signing certificate, or S/MIME certificate—and thus enables you to restrict the usage of a certificate to predetermined purposes.

- If the extension exists in a certificate, it limits the uses of the certificate to those specified (it limits the applications for a certificate).
- If the extension is not present, the certificate can be used for all applications except object signing.

The Netscape certificate type extension is a string of boolean bit-flags, each bit identifying the purpose for which a certificate to be used. Table 18.18 lists the bits and their designated purposes. The extension has no default value.

Table 18.18 Netscape certificate type extension bits and designated purposes

Bit	Purpose	Description
0	SSL Client	Specifies that the certificate can be used by clients for authentication during SSL connections.
1	SSL Server	Specifies that the certificate can be used by servers for authentication during SSL connections.
2	S/MIME	Specifies that the certificate can be used to send secure email messages.
3	Object Signing	Specifies that the certificate can be used for signing objects such as Java applets and plug-ins.
4	Reserved	This bit is reserved for future use.
5	SSL CA	Specifies that the certificate can be used by a CA to issue certificates for SSL connections.
6	S/MIME CA	Specifies that the certificate can be used by a CA to issue certificates for secure email.
7	Object Signing CA	Specifies that the certificate can be used by a CA to issue certificates for object signing.

The Netscape certificate type extension policy has been implemented in such a way that it enables you to set the appropriate certificate-type bits for certificates being issued by Certificate Management System. This way, you can restrict the purposes for which a certificate should be used by adding the extension, with the appropriate bits set, to the certificate at the time of issuance. For example, if you want to restrict a certificate to be used for SSL client authentication only, when issuing the certificate you would add the Netscape certificate type extension to the certificate with `ssl_client` (bit 0) set. For general guidelines on setting the Netscape certificate type extension, see "netscape-cert-type" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

In the current implementation, you can specify whether to add the extension to certificates on the server side and which bits in the extension are to be set on the client side—you specify whether to add the extension by enabling the Netscape certificate type extension policy and which bits are to be set by adding the appropriate HTTP variables to the enrollment forms.

Bits set in the Netscape certificate type extension are formed from pre-defined input variables that you can embed as hidden values in the default enrollment forms (see "Forms for Certificate Enrollment" on page 915). Table 18.19 lists the HTTP input variables that correspond to Netscape certificate type extension bits.

Table 18.19 HTTP input variables for Netscape certificate type extension bits

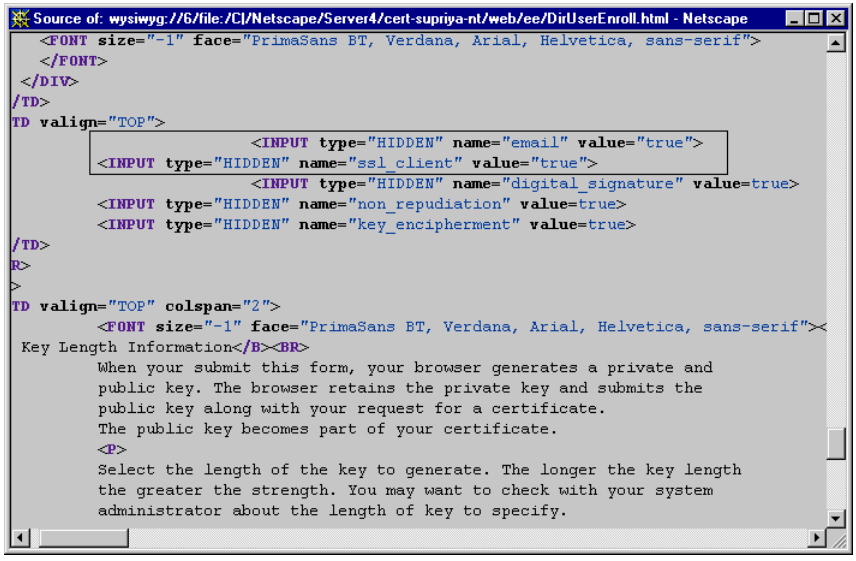
HTTP input variable	Netscape certificate type extension bit
`ssl_client`	SSL Client (bit 0)
`ssl_server`	SSL Server (bit 1)
`email`	S/MIME (bit 2)
`object_signing`	Object Signing (bit 3)
	Reserved for future use (bit 4)
`ssl_ca`	SSL CA (bit 5)
`email_ca`	S/MIME CA (bit 6)
`object_signing_ca`	Object Signing CA (bit 7)

During installation, Certificate Management System automatically creates an instance of the Netscape certificate type extension policy for the various types of certificates that you may want the server to issue. See “NSCertTypeExt Rule” on page 652.

Additionally, the default enrollment forms—the directory-based, directory- and PIN-based, manual, Kerberos server-based, and NIS server-based enrollment forms—for various types of certificates also include the appropriate HTTP input variables corresponding to Netscape certificate type extension bits. For details about these forms, see “Forms for Certificate Enrollment” on page 915.

Figure 18.20 shows the default directory-based enrollment form for end users with HTTP input variables specific to Netscape certificate type extension highlighted; it shows two of the total number of variables listed in Table 18.19, `ssl_client` and `email`, indicating that these bits be set in certificates requested using this form.

Figure 18.20 Netscape certificate type extension-specific variables in enrollment forms



Note that the default enrollment forms embed variables that are considered appropriate for the type of certificate, such as client, server, or CA, that can be requested using the form. For example, the server enrollment form embeds the `ssl_server` variable, whereas the subordinate CA (Certificate Manager) enrollment form embeds the `ssl_client`, `email_ca`, `ssl_ca` and `object_signing_ca` variables.

In general, the forms are set up so that you don't have to make any modifications. However, if there is a need to modify the bit settings, be sure to add or remove the corresponding variable. Also, when adding a new variable, make sure that the HTML input format is as follows:

```
<input type="HIDDEN" value="true" name="variable_name">
```

where *variable_name* can be any of the variables listed in Table 18.19.

NSCertTypeExt Module

The Java class that implements the Netscape certificate type extension policy is as follows:

```
com.netscape.certsrv.policy.NSCertTypeExt
```

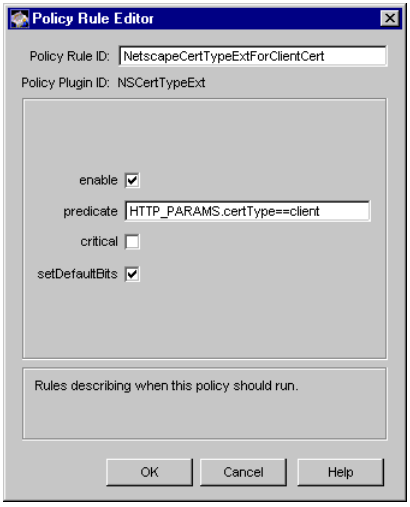
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.NSCertTypeExt.class=com.netscape.certsrv.policy.NSCertTypeExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `NSCertTypeExt`

Figure 18.21 shows how the configurable parameters for the `NSCertTypeExt` module are displayed in the CMS window.

Figure 18.21 Parameters defined in the NSCertTypeExt module



The configuration shown in Figure 18.21 creates a policy rule named `NetscapeCertTypeExtForClientCert`, which enforces a rule that the server should set the Netscape certificate type extension in client certificates.

Table 18.20 gives details about each of these parameters.

Table 18.20 Description of parameters defined in the NSCertTypeExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule, be sure to review the enrollment forms for Netscape certificate type extension-specific variables and to set the remaining parameters of this policy correctly. If the bits are unspecified in the enrollment form, the server checks the value assigned to the `setDefaultBits` parameter. If it is unchecked (false), the server does not set the extension. • If you disable the rule, the server does not add the extension to certificates; it ignores the Netscape certificate type extension-specific HTTP input values in the certificate request and the status of the `setDefaultBits` parameter.

Table 18.20 Description of parameters defined in the NSCertTypeExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`setDefaultBits`	Specifies whether to set the Netscape certificate type extension with default bits in certificates specified by the predicate expression. • Check the box the if you want the server to add the extension, with default bits, to certificates. If you check the box and if no bits are requested from the HTTP input, the server adds the Netscape certificate type extension to certificates with the following bits set: - `ssl client` (bit 0) - `email` (bit 2) • Uncheck the box if you don't want the server to add the extension with default bits. If you uncheck the box and if no bits are requested from the HTTP input, the server does not add the extension to certificates.

NSCertTypeExt Rule

The policy rule named `NSCertTypeExt` is an instance of the `NSCertTypeExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is set so that the extension gets added to all certificates except the ones issued to routers (`predicate=certType!=CEP-Request`).
- The server sets the default bits if the bits are unspecified in the enrollment form.

For details on individual parameters defined in the rule, see Table 18.20 on page 651. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

OCSP No Check Extension Policy

The `OCSPNoCheckExt` plug-in module implements the OCSP no check extension policy. This policy enables you to configure Certificate Management System to add the *OCSP No Check Extension* defined in X.509 and PKIX standard RFC 2560 (see `http://www.ietf.org/rfc/rfc2560.txt`) to certificates. The extension, which should be used in OCSP responder certificates only, indicates how OCSP-compliant applications can verify the revocation status of the certificate an authorized OCSP responder uses to sign OCSP responses.

The online certificate status protocol (OCSP) enables OCSP-compliant applications to determine the revocation status of a certificate being validated. Certificate Management System supports the OCSP service—you can configure a Certificate Manager to publish CRLs to an online validation authority (also called OCSP responder); see "Supported Methods for Verifying Revocation Status of Certificates" on page 724. If you configure Certificate Management System to work with an OCSP responder, OCSP-compliant applications in your PKI setup will be able to do real-time verification of certificates by querying the OCSP responder for their revocation status. Note that these applications will be able to query the OCSP responder only if the certificate being validated includes the authority information access extension indicating the location of the OCSP responder; for information on adding this extension to certificates, see "Authority Information Access Extension Policy" on page 558.

When queried by an application on the status of a certificate, the OCSP responder sends a digitally signed response. For the signature, the responder uses the key pair designated for signing OCSP responses. Usually, the CA issues an *OCSP responder certificate* to the responder, which enables applications to identify it as a CA-designated responder. The CA issues this certificate with an extended key usage extension with a unique value, which indicates that the key associated with the certificate can be used for signing OCSP responses. For details on this extension, see "OCSPSigningExt Rule" on page 603.

When an OCSP-compliant application receives a signed response, as a part of validating the signature, the application needs to verify that the responder's certificate has not been revoked. RFC 2560 recommends three ways in which a CA may indicate the revocation status of an OCSP responder certificate. One of them is that the CA issue the OCSP responder a certificate with the OCSP no check extension, which indicates that the certificate can be trusted by the

clients for its lifetime. The OCSP no check policy of Certificate Management System implements this method and enables you to set the OCSP no check extension in OCSP responder certificates.

Because OCSP-compliant applications don't check for the revocation status of the OCSP responder certificate (containing the OCSP no check extension), when issuing these types of certificates, you should consider issuing them with a short validity period (and renew them frequently). Note that the OCSP no check extension policy only adds the extension to a certificate; it doesn't control the validity period of the certificate. If you want to limit the validity period of these certificates to a short period, you should consider creating an instance of the `ValidityConstraints` module with the appropriate configuration, for example, set the predicate parameter to `HTTP_PARAMS.certType=ocspResponder`. For details, see "Validity Constraints Policy" on page 543. If you have agent privileges, you can also specify the required validity period when approving the OCSP responder certificate request in the request queue; the enrollment process for an OCSP responder certificate is manual, and the request gets queued for agent approval.

Before configuring the server to add the OCSP no check extension to OCSP responder certificates, read the general guidelines provided in "OCSPNocheck" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide.*

During installation, Certificate Management System automatically creates an instance of the OCSP no check extension policy. See "OCSPNoCheckExt Rule" on page 656.

OCSPNoCheck Module

The Java class that implements the OCSP no check extension policy is as follows:

`com.netscape.certsrv.policy.OCSPNoCheckExt`

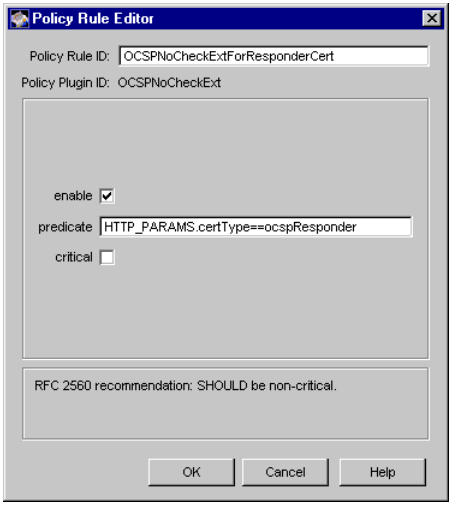
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.OCSPNoCheckExt.class=com.netscape.certsrv.policy.OCSPNoCheckExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `OCSPNoCheckExt`

Figure 18.22 shows how the configurable parameters for the `OCSPNoCheckExt` module are displayed in the CMS window.

Figure 18.22 Parameters defined in the OCSPNoCheckExt module



The configuration shown in Figure 18.22 creates a policy rule named `OCSPNoCheckExtForResponderCert`, which enforces a rule that the server should set the OCSP no check extension in OCSP responder certificates only.

Table 18.21 provides details for each of these parameters.

Table 18.21 Description of parameters defined in the OCSPNoCheckExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the OCSP no check extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.21 Description of parameters defined in the OCSPNoCheckExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ocspResponder`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

OCSPNoCheckExt Rule

The policy rule named `OCSPNoCheckExt` is an instance of the `OCSPNoCheckExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is set (`predicate=HTTP_PARAMS.certType==ocspResponder`) so that the extension gets added to OCSP responder certificates only.
- The extension is marked noncritical (to comply with the PKIX recommendation).

For details on individual parameters defined in the rule, see Table 18.21 on page 656. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Policy Constraints Extension Policy

The `PolicyConstraintsExt` plug-in module implements the policy constraints extension policy. This policy enables you to configure Certificate Management System to add the *Policy Constraints Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension, which can be used in CA certificates only, constrains path validation in two ways—either to prohibit policy mapping or to require that each certificate in a path contain an acceptable policy identifier.

The policy constraints extension policy in Certificate Management System allows setting of the policy constraints extension as defined in its X.509 definition. The policy allows you to specify both, `requireExplicitPolicy` and `inhibitPolicyMapping` fields. PKIX standard requires that, if present in a CA certificate, the extension must never consist of a null sequence. At least one of the two specified fields must be present. Before configuring the server to add the policy constraints extension to certificates, read the general guidelines provided in "policyConstraints" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

During installation, Certificate Management System automatically creates an instance of the policy constraints extension policy. See "PolicyConstraintsExt Rule" on page 661.

PolicyConstraintsExt Module

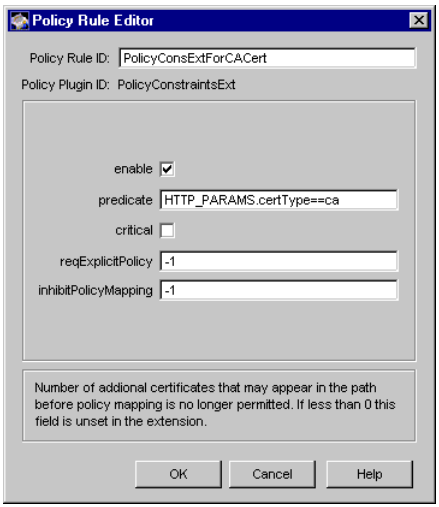
The Java class that implements the policy constraints extension policy is as follows:

```
com.netscape.certsrv.policy.PolicyConstraintsExt
```

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.PolicyConstraintsExt.class=com.netscape.certsrv.policy.PolicyConstraintsExt`
- In the CMS window, the module is identified as follows: `PolicyConstraintsExt`

Figure 18.23 shows how the configurable parameters for the `PolicyConstraintsExt` module are displayed in the CMS window.

Figure 18.23 Parameters defined in the PolicyConstraintsExt module



The configuration shown in Figure 18.23 creates a policy rule named `PolicyConsExtForCACert`, which enforces a rule that the server should set the policy constraints extension in CA certificates only.

Table 18.22 gives details about each of these parameters.

Table 18.22 Description of parameters defined in the PolicyConstraintsExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the policy constraints extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`

Table 18.22 Description of parameters defined in the PolicyConstraintsExt module (Continued)

Parameter	Description
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`reqExplicit Policy`	Specifies the total number of certificates permitted in the path before an explicit policy is required—that is, the number of CA certificates that can be chained below (subordinate to) the subordinate CA certificate being issued before an acceptable policy is required. Note that the number you specify affects the number of CA certificates to be used during certificate validation. The chain starts with the end-entity certificate being validated and moving up the chain. (The parameter has no effect if the extension is set in end-entity certificates.) Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension (default). • `0` specifies that no subordinate CA certificates are permitted in the path before an explicit policy is required. • `n` must be an integer that is greater than zero. It specifies at the most `n` subordinate CA certificates are allowed in the path before an explicit policy is required. Example: `1`
`inhibitPolicy Mapping`	Specifies the total number of certificates permitted in the path before policy mapping is no longer permitted. Permissible values: `-1`, `0`, or `n`. • `-1` specifies that the field should not be set in the extension (default). • `0` specifies that no subordinate CA certificates are permitted in the path before policy mapping is no longer permitted. • `n` must be an integer that is greater than zero. It specifies at the most `n` subordinate CA certificates are allowed in the path before policy mapping is no longer permitted. For example, a value of one indicates that policy mapping may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path. Example: `-1`

PolicyConstraintsExt Rule

The policy rule named `PolicyConstraintsExt` is an instance of the `PolicyConstraintsExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is disabled; for the rule to be effective, it must be enabled and configured appropriately.
- The predicate expression is set (`predicate=HTTP_PARAMS.certType==ca`) so that the extension gets added to CA certificates only. PKIX and Federal PKI standards recommend that CA certificates must have this extension and end-entity certificates should have this extension.
- The extension is marked noncritical.
- No subordinate CA certificates are permitted in the path before an explicit policy is required (`reqExplicitPolicy=0`).
- The `inhibitPolicyMapping` field is not set in the extension.

For details on individual parameters defined in the rule, see Table 18.22 on page 659. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Policy Mappings Extension Policy

The `PolicyMappingsExt` plug-in module implements the policy mappings extension policy. This policy enables you to configure Certificate Management System to add the *Policy Mappings Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension lists one or more pairs of OIDs, each pair identifying two policy statements of two CAs. The pairing indicates that the corresponding policies of one CA are equivalent to policies of another CA. The extension may be useful in the context of cross-certification.

The PKIX standard suggests that the extension must be marked noncritical and may be supported by CAs and/or applications. If supported, the extension is to be included in CA certificates only. Before configuring the server to add the

policy mappings extension to certificates, read the general guidelines provided in "policyMappings" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The policy mappings extension policy in Certificate Management System allows setting of the policy mappings extension as defined in its X.509 definition. The policy allows you to map policy statements of one CA to that of another by pairing the OIDs assigned to their policy statements. (For information on OIDs, see "Object Identifier" on page 553. For information on certificate policies, see "Certificate Policies Extension Policy" on page 574.)

Each pair is defined by two parameters, `issuerDomainPolicy` and `subjectDomainPolicy`. The pairing indicates that the issuing CA considers the `issuerDomainPolicy` equivalent to the `subjectDomainPolicy` of the subject CA. The issuing CA's users may accept an `issuerDomainPolicy` for certain applications. The policy mapping tells these users which policies associated with the subject CA are equivalent to the policy they accept.

During installation, Certificate Management System automatically creates an instance of the policy mappings extension policy. See "PolicyMappingsExt Rule" on page 665.

PolicyMappingsExt Module

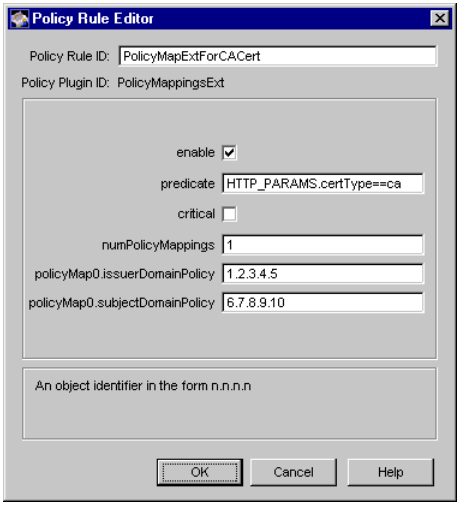
The Java class that implements the policy mappings extension policy is as follows:

```
com.netscape.certsrv.policy.PolicyMappingsExt
```

- In the CMS configuration file, the module is identified as follows: `ca.Policy.impl.PolicyMappingsExt.class=com.netscape.certsrv.policy.PolicyMappingsExt`
- In the CMS window, the module is identified as follows: `PolicyMappingsExt`

Figure 18.24 shows how the configurable parameters for the `PolicyMappingsExt` module are displayed in the CMS window.

Figure 18.24 Parameters defined in the PolicyMappingsExt module



The configuration shown in Figure 18.24 creates a policy rule named `PolicyMapExtForCACert`, which enforces a rule that the server should set the policy mappings extension in CA certificates only.

Table 18.23 provides details for each of these parameters.

Table 18.23 Description of parameters defined in the PolicyMappingsExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the policy mappings extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`

Table 18.23 Description of parameters defined in the PolicyMappingsExt module (Continued)

Parameter	Description
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`numPolicyMappings`	Specifies the total number of policy mapping (pairs) to be contained or allowed in the extension. Note that each policy mapping represents a pair of policies—specified by `policyMap<n>.issuerDomainPolicy` and `policyMap<n>.subjectDomainPolicy`—and each policy in the pair belongs to a specific CA. You can change the total number of policy pairs by changing the value assigned to this parameter; there's no restriction on the total number of policy pairs you can include in the extension. Each pair is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numPolicyMappings` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no policy pairs can be contained in the extension. • `n` specifies the total number of policy pairs to be included in the extension; it must be a integer greater than zero. The default value is `1`. Example: `2`
`policyMap<n>. issuerDomainPolicy`	Specifies the OID assigned to the policy statement`<n>` of the issuing CA that you want to map with the policy statement of another CA. Permissible values: Any valid OID specified in dot-separated numeric component notation (see the example). The OID that you specify should be in the registered subtree of IDs reserved for your company's use. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.5`

Table 18.23 Description of parameters defined in the PolicyMappingsExt module (Continued)

Parameter	Description
`policyMap<n>. subjectDomainPolicy`	Specifies the OID assigned to the policy statement`<n>` of the subject CA that corresponds to the policy statement of the issuing CA. Permissible values: Any valid OID specified in dot-separated numeric component notation (see the example). Example: `6.7.8.9.10`

PolicyMappingsExt Rule

The rule named `PolicyMappingsExt` is an instance of the `PolicyMappingsExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is set (`predicate=HTTP_PARAMS.certType==ca`) so that the extension gets added to CA certificates only.
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The number of policy mappings is set to 1 (`numPolicyMappings=1`) indicating that a pair of policies are to be mapped.
- The fields for entering the OIDs for policies that are to be mapped are left blank for you to enter the appropriate values.

For details on individual parameters defined in the rule, see Table 18.23 on page 663. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Private Key Usage Period Extension Policy

The `PrivateKeyUsagePeriodExt` plug-in module implements the private key usage period extension policy. This policy enables you to configure Certificate Management System to add the *Private Key Usage Period Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension allows the certificate issuer to specify a different validity period for the private key than the one specified for the corresponding certificate. The extension is intended for use with digital signature keys.

The PKIX standard recommends against the use of this extension. The standard also recommends that CAs conforming to the standard must not generate certificates with private key usage period extensions that are marked critical. For general guidelines on setting this extension in certificates, see "privateKeyUsagePeriod" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The private key usage period extension policy in Certificate Management System allows setting of the private key usage period extension as defined in its X.509 definition. The policy enables you to specify values for the `notBefore` and `notAfter` components. When included in a certificate, the `notBefore` and `notAfter` components define the time before and after which the private key associated with the certificate should not be used to sign objects.

PrivateKeyUsagePeriodExt Module

The Java class that implements the private key usage period extension policy is as follows:

```
com.netscape.certsrv.policy.PrivateKeyUsagePeriodExt
```

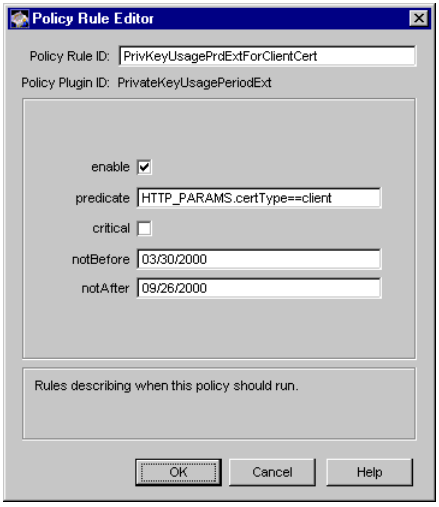
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.PrivateKeyUsagePeriodExt.class=com.netscape.certsrv.policy.PrivateKeyUsagePeriodExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `PrivateKeyUsagePeriodExt`

Figure 18.25 shows how the configurable parameters for the `PrivateKeyUsagePeriodExt` module are displayed in the CMS window.

Figure 18.25 Parameters defined in the PrivateKeyUsagePeriodExt module



The configuration shown in Figure 18.25 creates a policy rule named `PrivKeyUsagePrdExtForClientCert`, which enforces a rule that the server should set the private key usage period extension in client certificates.

Table 18.24 provides details for each of these parameters.

Table 18.24 Description of parameters defined in the PrivateKeyUsagePeriodExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the private key usage period extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.24 Description of parameters defined in the PrivateKeyUsagePeriodExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`notBefore`	Specifies the date on which the validity period for the private key associated with the certificate begins. Permissible values: A valid date specified in the `MM/DD/YYYY` format. Example: `12/25/2000`
`notAfter`	Specifies the date on which the validity period for the private key associated with the certificate ends. Permissible values: A valid date specified in the `MM/DD/YYYY` format. Example: `12/25/2001`

Subject Alternative Name Extension Policy

The `SubjectAltNameExt` plug-in module implements the subject alternative name policy. This policy enables you to configure Certificate Management System to add the *Subject Alternative Name Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension enables you to bind additional identities—such as Internet electronic mail address, a DNS name, an IP address, and a uniform resource indicator (URI)—to the subject of the certificate.

The standard suggests that if the certificate subject field contains an empty sequence, then the subject alternative name extension must contain the subject's alternative name and that the extension be marked critical. For general

guidelines on setting the subject alternate name extension in certificates, see "subjectAltName" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The subject alternative name extension policy in Certificate Management System enables you to include values of certificate-request attributes in the extension. You can include any number of attributes as long as the attribute values conform to any of the supported general-name forms: rfc822Name, X.500 directory name, DNS name, EDI party name, URL, IP address, object identifier, and Other name.

Attributes in a certificate request are filled in by servlets from the HTTP input forms used for request submission. Some attributes, such as passwords typed in the form are not stored in the request. Other attributes regarding the end entity, such as the user ID, are set on the request after successful authentication. The servlets can also set additional attributes related to the certificate content on the request; for example, in automated-enrollment methods, some attributes may be read from the authentication directory and set in the request as authenticated attributes.

If you're using any of the directory-based authentication methods, you can configure Certificate Management System to retrieve values for any string and byte attributes from the directory and set them in the certificate request during authentication—you specify these attributes by entering them in the `ldapStringAttributes` and `ldapByteAttributes` fields defined in the automated enrollment modules. For more information, see Table 10.2 on page 329, Table 10.3 on page 334, and Table 10.4 on page 344.

Note that all data related to an end entity is gathered at the servlet level and set on the request before the request is passed to the policy subsystem.

In general, you can configure which attributes should or shouldn't be stored in the request; for example, you can exclude sensitive attributes such as passwords from getting stored in the request with the help of the parameter named `dontSaveHttpParams` defined in the CMS configuration file. For details on using this parameter, see the description for `HTTP_PARAMS` in Table 24.9 on page 924. You can also distinguish the attributes based on their origin—that is, whether they originated from the enrollment form or where added to the request during the authentication process. Authenticated attributes have `AUTH_TOKEN` as prefix (for example, `AUTH_TOKEN.mail`) and non-authenticated attributes such as the ones that come from the HTTP input have `HTTP_PARAMS` as prefix (for example, `HTTP_PARAMS.csrRequestorEmail`).

If enabled, the subject alternative extension policy checks the certificate request for configured attributes. If the request contains an attribute, the policy reads its value and sets it in the extension. This way, the extension that gets to added to certificates contains all the configured attributes.

During installation, Certificate Management System automatically creates an instance of the subject alternative name extension policy. See "SubjectAltNameExt Rule" on page 673.

SubjectAltNameExt Module

The Java class that implements the subject alternate name extension policy is as follows:

`com.netscape.certsrv.policy.SubjectAltNameExt`

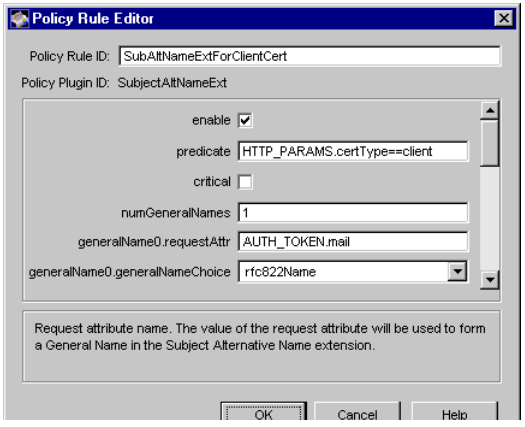
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.SubjectAltNameExt.class=com.netscape.certsrv.policy.SubjectAltNameExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `SubjectAltNameExt`

Figure 18.26 shows how the configurable parameters for the `SubjectAltNameExt` module are displayed in the CMS window.

Figure 18.26 Parameters defined in the SubjectAltNameExt module



The configuration shown in Figure 18.26 creates a policy rule named `SubAltNameExtForClientCert`, which enforces a rule that the alternative name of the certificate's subject must be derived from the `mail` attribute of subject's entry in the authentication directory and that the server should set the subject alternative name extension only in client certificates.

Table 18.25 provides details for each of these parameters.

Table 18.25 Description of parameters defined in the SubjectAltNameExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server adds the subject alternative name extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

Table 18.25 Description of parameters defined in the SubjectAltNameExt module (Continued)

Parameter	Description
`numGeneralNames`	Specifies the total number of alternative names or identities permitted in the extension. Note that each name has a set of configuration parameters—`generalName<n>.requestAttr` and `generalName<n>.generalNameChoice`—and you must specify appropriate values for each of those parameters; otherwise the policy rule will return an error. You can change the total number of identities by changing the value of this parameter; there's no restriction on the total number of identities you can include in the extension. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numGeneralNames` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `0` or `n`. • `0` specifies that no identities can be contained in the extension. • `n` specifies the total number of identities to be included in the extension; it must be an integer greater than zero. The default value is 8. Example: `2`
`generalName<n>.requestAttr`	Specifies the request attribute whose value is to be included in the extension. The attribute value must conform to any of the supported general-name types (specified by the `generalName<n>.generalNameChoice` parameter). If the server finds the attribute in the request, it sets the attribute value in the extension and then adds the extension to certificates specified by the `predicate` parameter. If you specify multiple attributes and if none of the attributes are present in the request, the server does not add the subject alternative name extension to certificates. Permissible values: A request attribute included in the certificate request. Example: `AUTH_TOKEN.mail`

Table 18.25 Description of parameters defined in the SubjectAltNameExt module (Continued)

Parameter	Description
`generalName<n>.generalNameChoice`	Specifies the general-name type for the request attribute. Permissible values: `rfc822Name`, `directoryName`, `dNSName`, `ediPartyName`, `URL`, `iPAddress`, `OID`, or `otherName`. • Select `rfc822Name` if the request-attribute value is an Internet mail address in the `local-part@domain` format (default). For example, `jdoe@siroe.com`. • Select `directoryName` if the request-attribute value is an X.500 directory name, similar to the subject name in a certificate. For example, `CN=Jane Doe, OU=Sales Dept, O=Siroe Corp, C=US`. • Select `dNSName` if the request-attribute value is a DNS name. For example, `corpDirectory.siroe.com`. • Select `ediPartyName` if the request-attribute value is a EDI party name. For example, `Siroe Corporation`. • Select `URL` if the request-attribute value is a non-relative URI that includes both a scheme (for example, `http`) and a fully qualified domain name or IP address of the host. For example, `http://hr.siroe.com`. • Select `iPAddress` if the request-attribute value is a valid IP address specified in dot-separated numeric component notation. For example, `128.21.39.40`. • Select `OID` if the request-attribute value is a unique, valid OID specified in the dot-separated numeric component notation. For example, `1.2.3.4.55.6.5.99`. • Select `otherName` if the request-attribute value is the absolute path to the file that contains the base-64 encoded string of the subject alternative name. For example, `/usr/netscape/server4/ext/san/othername.txt`. Example: `rfc822Name`

SubjectAltNameExt Rule

The policy rule named `SubjectAltNameExt` is an instance of the `SubjectAltNameExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.

- The predicate expression is left blank so that the extension gets added to all certificates the server issues. (PKIX and Federal PKI standards recommend that CA certificates must have this extension and end-entity certificates should have this extension.)
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The rule is configured to include at the most three alternative names in the extension (`numGeneralNames=3`).
- The first alternative name is the value of the `mail` attribute in the certificate subject's directory entry (`generalName0.requestAttr=AUTH_TOKEN.mail`) and the name is in the `rfc822Name` format (`generalName0.generalNameChoice=rfc822Name`).
- The second alternative name is the value of the `mailalternateaddress` attribute in the certificate subject's directory entry (`generalName1.requestAttr=AUTH_TOKEN.mailalternateaddress`) and the name is in the `rfc822Name` format (`generalName1.generalNameChoice=rfc822Name`).
- The third alternative name is the value of an HTTP input parameter `csrRequestorEmail` included in the certificate request (`generalName2.requestAttr=HTTP_PARAMS.csrRequestorEmail`) and the name is in `rfc822Name` format (`generalName2.generalNameChoice=rfc822Name`).

For details on individual parameters defined in the rule, see Table 18.25 on page 671. You need to review this rule and make the changes appropriate for your PKI setup. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. For instructions on adding additional instances, see "Step 4. Add New Policy Rules" on page 697.

Before you edit the default rule, you should read the additional details about the attributes that are set in the default policy rule.

The first two attributes, `AUTH_TOKEN.mail` and `AUTH_TOKEN.mailalternateaddress`, are standard LDAP attributes typically used for storing end users' email addresses in an LDAP directory. These attributes enable you to include a user's email address as an alternative name in the certificate. Remember that you need to specify the LDAP attribute for users' email addresses as a part of configuring the server to use a specific directory for

authentication—which means for the default rule to set end users' email addresses in the subject alternative name extension, you must ensure the following:

- The server is configured for directory-based, directory- and PIN-based, or NIS server based (using directory attributes for forming subject names) enrollment; that is, you have created and configured an authentication instance.
- The `ldapStringAttributes` parameter in the authentication instance is set to `mail` or `mailalternateaddress`, or to both.

The third attribute, `HTTP_PARAMS.csrRequestorEmail`, is the email component of the subject name in an enrollment request—it is an HTTP input value that gets added to the request when a user uses the *manual* enrollment form; for details, see "Enrollment Forms" on page 361.

If you enable the default policy rule, the server automatically checks the certificate request for attributes `AUTH_TOKEN.mail`, `AUTH_TOKEN.mailalternateaddress`, and `HTTP_PARAMS.csrRequestorEmail`. If the server finds any of the attributes, it sets the attribute value in the extension and then adds the extension to certificates specified by the `predicate` parameter. If none of the attributes are in a request, the server does not add the subject alternative name extension to the certificate.

Subject Directory Attributes Extension Policy

The `SubjectDirectoryAttributesExt` plug-in module implements the subject directory attributes extension policy. This policy enables you to configure Certificate Management System to add the *Subject Directory Attributes Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension is used to specify any desired directory attribute values for the subject of the certificate.

As per the PKIX standard, inclusion of this extension in certificates is not essential; the standard suggests that the extension may be used in local environments. For general guidelines on setting the subject directory attributes extension, see "subjectDirectoryAttributes" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The subject directory attributes extension policy in Certificate Management System allows you to include up to three directory attributes in the extension. For each attribute that you want to include in the extension, you need to specify the attribute name and its value—the name must be the X.500 directory attribute name itself and the attribute value can be derived from the request or directly entered in the policy configuration as a string value.

The list of directory attributes supported by default are shown as permissible values for the `attribute<n>.attributeName` parameter explained in Table 18.26 on page 677. You can extend the list of attributes supported by the policy by defining new X.500 directory attributes. For details on defining new attributes, see "Extending Attribute Support" on page 1160.

Note that, during installation, Certificate Management System does not create an instance of the subject directory attributes extension policy. If you want the server to add this extension to certificates, you must create an instance of the `SubjectDirectoryAttributesExt` module and configure it. For instructions, see "Step 4. Add New Policy Rules" on page 697.

SubjectDirectoryAttributesExt Module

The Java class that implements the subject directory attributes extension policy is as follows:

```
com.netscape.certsrv.policy.SubjectDirectoryAttributesExt
```

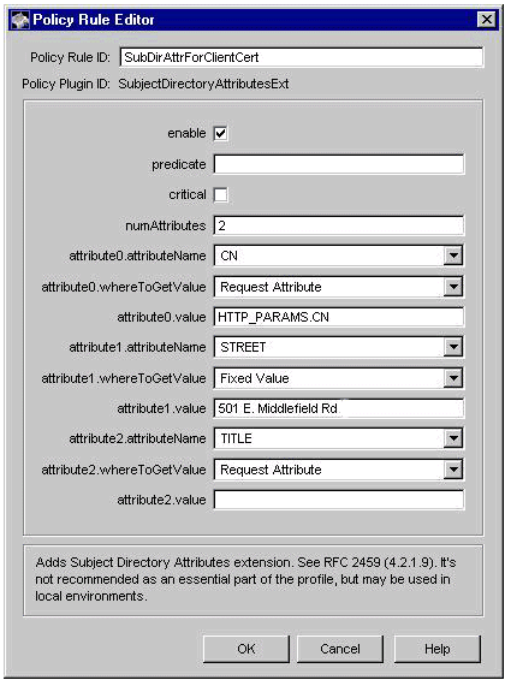
- In the CMS configuration file, the module is identified as follows:
  `<subsystem>.Policy.impl.SubjectDirectoryAttributesExt.class=`
  `com.netscape.certsrv.policy.SubjectDirectoryAttributesExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows:
  `SubjectDirectoryAttributesExt`

Figure 18.27 shows how the configurable parameters for the `SubjectDirectoryAttributesExt` module are displayed in the CMS window.

Figure 18.27 Parameters defined in the SubjectDirectoryAttributesExt module



The configuration shown in Figure 18.27 creates a policy rule named `SubDirAttrForClientCert`, which enforces a rule that the server should set the subject directory attributes extension in client certificates.

Table 18.26 provides details for each of these parameters.

Table 18.26 Description of parameters defined in the SubjectDirectoryAttributesExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the subject directory attributes extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.

Table 18.26 Description of parameters defined in the SubjectDirectoryAttributesExt module (Continued)

Parameter	Description
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==client AND HTTP_PARAMS.OU==Engineering`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`numAttributes`	Specifies the total number of directory attributes to be contained or allowed in the extension. Note that each attribute has a name (or OID) and value and you must specify appropriate values for both; otherwise the policy rule will return an error. You can configure the server to include up to three attributes in the extension. By default, this field is set to its maximum value, `3`, and the UI shows fields for configuring three attributes. You can change the total number of attributes by changing the value of this parameter. Each set of configuration parameters is distinguished by `<n>`, which is an integer derived from the value you assign in this field. For example, if you set the `numAttributes` parameter to 2, `<n>` would be `0` and `1`. Permissible values: `1`, `2`, or `3`. The default value is 3. Example: `1`
`attribute<n>.attributeName`	Specifies the name of the directory attribute whose value is to be included in the extension. Permissible values: `TITLE`, `O`, `OU`, `L`, `E`, `C`, `GIVENNAME`, `DC`, `UID`, `CN`, `UNSTRUCTUREDNAME`, `GENERATIONQUALIFIER`, `ST`, `DNQUALIFIER`, `SN`, `MAIL`, `UNSTRUCTUREDADDRESS`, `STREET`, `SERIALNUMBER`, and `INITIALS`. The list may show any additional attributes that you may have added. Example: `TITLE`

Table 18.26 Description of parameters defined in the SubjectDirectoryAttributesExt module (Continued)

Parameter	Description
`attribute<n>.whereToGetValue`	Specifies from where to get the value for the selected directory attribute. Permissible values: `Request Attribute` or `Fixed Value`. • Select `Request Attribute` if you want the server to read the value from the request attribute. • Select `Fixed Value` if you want to specify a fixed value for the attribute. Note that both the options require you to enter the value for the attribute in the `attribute<n>.value` field. The server will set the extension with this value in all certificates specified by the `predicate` parameter. Example: `Fixed Value`
`attribute<n>.value`	Specifies the value for the directory attribute to be included in the extension. Permissible value: A string value for the attribute selected. Example: `Member of Technical Staff`

Subject Key Identifier Extension Policy

The `SubjectKeyIdentifierExt` plug-in module implements the subject key identifier policy. This policy enables you to configure Certificate Management System to add the *Subject Key Identifier Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to certificates. The extension is used to identify certificates that contain a particular public key—that is, the extension is used to uniquely identify a certificate from among several that have the same subject name.

Typically, the subject key identifier extension is used in CA certificates as it helps determine which CA key is being certified in a CA certificate. To facilitate chain building, you should consider adding this extension to conforming subordinate CA certificates (subordinate Certificate Managers' CA signing certificates) issued by Certificate Management System. You may also want to consider adding this extension to other or all certificates. For example, if added to end-entity certificates, the extension provides a means for identifying certificates containing the particular public key used in an application. If an end

entity has multiple certificates, especially from multiple CAs, the subject key identifier provides a means to quickly identify the set of certificates that contain a particular public key.

For general guidelines on setting the subject key identifier extension, see "subjectKeyIdentifier" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The subject key identifier extension policy in Certificate Management System allows setting of the subject key identifier extension as defined in its X.509 definition. It enables you to specify the method for forming the Key Identifier.

By default, the policy supports three types of methods for deriving the Key Identifier; the default methods for forming the Key Identifier are based on PKIX recommendations as defined in section 4.2.1.2. They are as follows:

- 20 byte (160 bit) SHA-1 hash of the BIT STRING of Subject Public Key.
- A type field value of 0100 followed by 60 least significant bits of the SHA-1 hash of the Subject Public Key.
- 20 byte (160 bit) SHA-1 hash of the Subject Public Key Info. This is how Netscape Communicator generates a Key Identifier (but is not necessary to be compatible with the Communicator).

You can also customize the method for deriving the Key Identifier by subclassing the policy and overriding the following method:

```
formKeyIdentifier(X509CertInfo certInfo, IRequest req)
```

For details, check the CMS SDK installed at this location:

```
<server_root>/cms_sdk/sdkdocs
```

You may also want to check the CMS samples installed here:

```
<server_root>/cms_sdk/samples
```

If enabled, the policy adds a Subject Key Identifier Extension to an enrollment request if the extension does not already exist. If the extension exists in the request, for example from a CRMF request, the policy replaces the extension. In case of manual enrollments, after an agent approves the enrollment request, the policy accepts any Subject Key Identifier Extension that is already there.

During installation, Certificate Management System automatically creates an instance of the subject key identifier extension policy. See “SubjectKeyIdentifierExt Rule” on page 683.

SubjectKeyIdentifierExt Module

The Java class that implements the subject key identifier extension policy is as follows:

`com.netscape.certsrv.policy.SubjectKeyIdentifierExt`

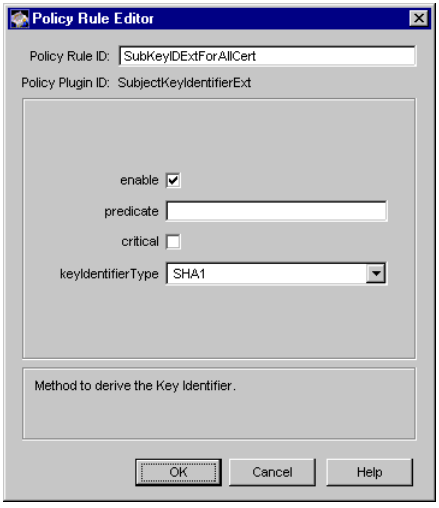
- In the CMS configuration file, the module is identified as follows: `<subsystem>.Policy.impl.SubjectKeyIdentifierExt.class=com.netscape.certsrv.policy.SubjectKeyIdentifierExt`

  where `<subsystem>` is `ca` or `ra` (prefix identifying the subsystem)

- In the CMS window, the module is identified as follows: `SubjectKeyIdentifierExt`

Figure 18.28 shows how the configurable parameters for the `SubjectKeyIdentifierExt` module are displayed in the CMS window.

Figure 18.28 Parameters defined in the SubjectKeyIdentifierExt module



The configuration shown in Figure 18.28 creates a policy rule named `SubKeyIDExtForAllCert`, which enforces a rule that the server should set the subject key identifier extension in all certificates.

Table 18.27 provides details for each of these parameters.

Table 18.27 Description of configuration parameters defined in the SubjectKeyIdentifierExt module

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server adds the subject key identifier extension to certificates specified by the `predicate` parameter. • If you disable the rule, the server does not add the extension to certificates; it ignores the values in the remaining fields.
`predicate`	Specifies the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490. Example: `HTTP_PARAMS.certType==ca`
`critical`	Specifies whether the extension should be marked critical or noncritical in certificates specified by the `predicate` parameter. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).
`KeyIdentifierType`	Specifies the method for deriving Key Identifier. Permissible values: `SHA1`, `TypeField`, or `SpkiSHA1`. • `SHA1` specifies that the key identifier must be derived as a 20 byte (160 bit) SHA-1 hash of the BIT STRING of Subject Public Key (default). • `TypeField` specifies that the key identifier must be derived as a type field value of 0100 followed by 60 least significant bits of the SHA-1 hash of the Subject Public Key. • `SpkiSHA1` specifies that the key identifier must be derived as a 20 byte (160 bit) SHA-1 hash of the Subject Public Key Info. Example: `SHA1`

SubjectKeyIdentifierExt Rule

The policy rule named `SubjectKeyIdentifierExt` is an instance of the `SubjectKeyIdentifierExt` module. Certificate Management System automatically creates this rule during installation. By default, the rule is configured as follows:

- The rule is enabled.
- The predicate expression is left blank so that the extension gets added to all certificates the server issues. (PKIX and Federal PKI standards recommend that CA certificates must have this extension and end-entity certificates should have this extension.)
- The extension is marked noncritical (to comply with the PKIX recommendation).
- The key identifier is a 20 byte (160 bit) SHA-1 hash of the BIT STRING of Subject Public Key (`KeyIdentifierType=SHA1`).

For details on individual parameters defined in the rule, see Table 18.27 on page 682. It is important that you review this rule and make the appropriate changes required by your PKI setup. For example, if you're planning to issue multiple certificates to an end entity and want to assist applications in identifying the appropriate end-entity certificate, you should consider modifying the predicate expression to add this extension to all end-entity certificates. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691. Also, if you need to add additional instances, follow the instructions in "Step 4. Add New Policy Rules" on page 697.

Chapter

19

Configuring a Subsystem's Policies

Netscape Certificate Management System (CMS) provides a customizable policy framework for its main subsystems, the Certificate Manager, Registration Manager, and Data Recovery Manager. This chapter explains how to configure these subsystems to apply organizational and other policies on incoming certificate and key-related requests.

Before reading this chapter, you should have read the previous chapters in this part. In particular, you should be familiar with the various policy plug-in modules that come with Certificate Management System. If you are not, see "Constraints-Specific Policy Modules" on page 501 and "Extension-Specific Policy Modules" on page 549.

The chapter has the following sections:

Policy Management

You can manage both constraints and extensions policy rules in two ways:

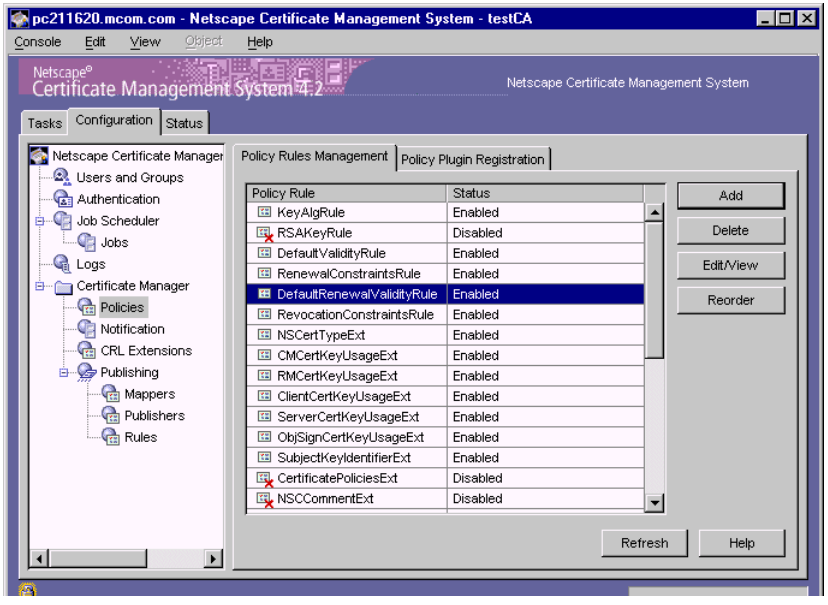
- By making changes to the authentication-specific parameters from the CMS window, explained in “Policy Management From the CMS Window” on page 686
- By editing the parameters in the configuration file, explained in “Policy Parameters in the Configuration File” on page 689

The recommended method is to use the CMS window.

Policy Management From the CMS Window

The CMS window (as shown in Figure 19.1) provides the appropriate user interface to support policy management for each subsystem, the Certificate Manager, Registration Manager, and Data Recovery Manager.

Figure 19.1 Policy information in the CMS window



Under each subsystem tree node in the CMS window, you will find a `Policies` object. This object represents the policy configuration of that subsystem, enabling you to accomplish the following operations:

- Configuration of currently registered policy modules
- Registration of new (custom) policy modules

The sections that follow describe the parts of the window from which you carry out these operations.

Policy Rules Management Tab

The Policy Rules Management tab displays the current policy configuration for the selected subsystem. The tab lists the currently configured policy instances (or rules) for a subsystem, enabling you to manage them at a single place. From this tab you can add, modify, or delete rules, enable or disable individual rules, and change the order in which the rules get applied to an end-entity request.

Add. The add operation shows a list of registered policy modules from which you can select the one you want to configure. When you save the changes, the subsystem creates the rule and displays it in the list of policy rules. For instructions, see "Step 4. Add New Policy Rules" on page 697.

Delete. The delete operation allows you to remove unwanted policy rules from the CMS configuration. For instructions, see "Step 3. Delete Unwanted Policy Rules" on page 697.

Edit/View. The edit operation allows you to view and modify configuration parameter values associated with the currently configured policy rules. For instructions, see "Step 2. Modify Existing Policy Rules" on page 691.

Reorder. The reorder operation allows you to change the order of the policy rules a subsystem applies to an end-entity request. Enabled rules are applied in the order in which they appear; disabled rules are ignored. For instructions, see "Step 5. Reorder Policy Rules" on page 705.

Policy Plugin Registration Tab

The Policy Plugin Registration tab lists the currently registered policy plug-in modules for the selected subsystem and gives you access to the window from which you can register new modules. On this tab you will find the names of registered modules listed on the left and the path to the Java class that implements the module listed on the right.

You can perform the following operations from this tab:

Register. This operation allows you to register a new policy module. For instructions, see "Registering a Policy Module" on page 709.

Delete. This operation allows you to remove unwanted policy modules from the CMS framework. For instructions, see "Deleting a Policy Module" on page 711.

Policy Parameters in the Configuration File

The sample configuration file shown on page 90 illustrates how policy-specific information appears in the configuration file. Keep the following points in mind:

- All policy-specific information, such as registered policy plug-in implementations, configured rules, and ordering, appear in the Policy section of the configuration file. If you have installed more than one subsystem in a CMS instance, for example Certificate Manager and Data Recovery Manager together, the configuration file will include policy sections that are specific to each of the subsystems that share the configuration.

  You can identify policy pertaining to a subsystem by these prefixes:

  — Certificate Manager by `ca`

  — Registration Manager by `ra`

  — Data Recovery Manager by `kra`

- Each registered policy plug-in module is identified by its implementation name and the corresponding Java class.

- Each configured rule of a policy module is identified by the name specified when the rule was created.
- You can create multiple rules out of an implementation; each rule must have a unique name.

To change the configuration by editing the configuration file, follow the instructions in "Changing the Configuration by Editing the Configuration File" on page 86.

Setting up Policy Rules for a Subsystem

You can configure the main subsystems of Certificate Management System (CMS)—the Certificate Manager, Registration Manager, and Data Recovery Manager—to apply certain organizational policies on end entities' certificate enrollment, renewal, and revocation requests before servicing them. This section explains how to configure a subsystem to evaluate end-entity requests based on a set of policy rules. The steps are as follows:

- Step 1. Plan
- Step 2. Modify Existing Policy Rules
- Step 3. Delete Unwanted Policy Rules
- Step 4. Add New Policy Rules
- Step 5. Reorder Policy Rules
- Step 6. Restart the Server
- Step 7. Test Policy Configuration

For information on adding or changing policy-specific information in the configuration file, see "Policy Parameters in the Configuration File" on page 689.

Step 1. Plan

Before configuring a Certificate Manager's or Registration Manager's policy, be sure to do this:

- Refer to the X.509 standard and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) to get familiar with certificate content, including extensions.
- Read chapters "Constraints-Specific Policy Modules" on page 501 and determine the rules that you want to use to govern the generation and formulation of certificates in your PKI setup.
- Read "Extension-Specific Policy Modules" on page 549 to determine the extensions you want to set in certificates.

This planning will help you configure a Certificate Manager and Registration Manager with the appropriate policy rules so that your end entities get the right kind of certificate.

Step 2. Modify Existing Policy Rules

You can modify a policy rule by editing its configuration parameter values; you cannot edit the name of a rule. To change the name of a rule, you need to create a new rule exactly like the rule you want to rename, except with a new name, and delete the old rule.

As a part of editing a rule, you can change its status from enabled to disabled or vice versa by checking or unchecking the `enable` parameter. A subsystem subjects certificate requests only to those rules that are enabled.

During installation, the Certificate Manager and Registration Manager automatically create a set of policy rules (that you would most likely want to use) using the policy modules registered by default. Figure 19.2 shows the policy rules created for a Certificate Manager. The Registration Manager also has a similar list. Table 19.1 summarizes the default rules created for both Certificate Manager and Registration Manager.

After installation, you must verify whether you want to use these rules, check how these rules are configured, and make the appropriate configuration changes. Keep in mind some of these policy rules are essential for the server to

process requests. For example, the server won't be able to process certificate-issuance requests if the *validity constraints* rule is disabled; see "Validity Constraints Policy" on page 543. Similarly, the server won't be able to process certificate-renewal requests if the *renewal validity* constraints rule is disabled; see "Renewal Validity Constraints Policy" on page 525.

If you don't want to use a rule, delete it from the configuration as explained in "Step 3. Delete Unwanted Policy Rules" on page 697; alternatively, you may keep it disabled. If you want to create a new rule, you can do so as explained "Step 4. Add New Policy Rules" on page 697.

Figure 19.2 Default policy rules created for a Certificate Manager

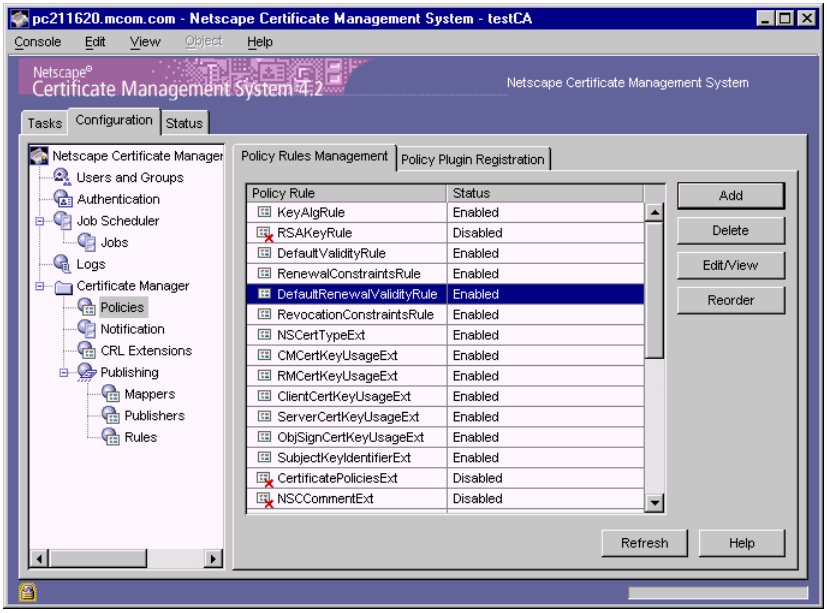


Table 19.1 Default policy rules created for a Certificate Manager and Registration Manager

Policy rule name	Created for Certificate Manager	Created for Registration Manager
`KeyAlgRule` See "KeyAlgRule Rule" on page 514.	Yes	Yes
`DSAKeyRule` Created only if the CA signing key is DSA. See "DSAKeyRule Rule" on page 508.	Yes	Yes
`RSAKeyRule` See "RSAKeyRule Rule" on page 532.	Yes	Yes
`DefaultValidityRule` See "DefaultValidityRule Rule" on page 548.	Yes	Yes
`RenewalConstranitsRule` See "RenewalConstraintsRule Rule" on page 521.	Yes	Yes
`DefaultRenewalValidityRule` See "DefaultRenewalValidityRule Rule" on page 528.	Yes	Yes
`RevocationConstranitsRule` See "RevocationConstraintsRule Rule" on page 524.	Yes	Yes
`NSCertTypeExt` See "NSCertTypeExt Rule" on page 652.	Yes	Yes
`CMCertKeyUsageExt` See "CMCertKeyUsageExt Rule" on page 626.	Yes	Yes
`RMCertKeyUsageExt` See "RMCertKeyUsageExt Rule" on page 627.	Yes	Yes
`ClientCertKeyUsageExt` See "ClientCertKeyUsageExt Rule" on page 629.	Yes	Yes

Table 19.1 Default policy rules created for a Certificate Manager and Registration Manager (Continued)

Policy rule name	Created for Certificate Manager	Created for Registration Manager
`ServerCertKeyUsageExt` See "ServerCertKeyUsageExt Rule" on page 628.	Yes	Yes
`ObjSignCertKeyUsageExt` See "ObjSignCertKeyUsageExt Rule" on page 631.	Yes	Yes
`SubjectKeyIdentifierExt` See "SubjectKeyIdentifierExt Rule" on page 683.	Yes	Yes
`CertificatePoliciesExt` See "CertificatePoliciesExt Rule" on page 580.	Yes	Yes
`NSCCommentExt` See "NSCCommentExt Rule" on page 646.	Yes	Yes
`OCSPNoCheckExt` See "OCSPNoCheckExt Rule" on page 656.	No	No
`OCSPSigningExt` See "OCSPSigningExt Rule" on page 603.	Yes	Yes
`CODESigningExt` See "CODESigningExt Rule" on page 602.	Yes	Yes
`CRLDistributionPointsExt` See "CRLDistributionPointsExt Rule" on page 597.	Yes	Yes
`AuthorityKeyIdentifierExt` See "AuthorityKeyIdentifierExt Rule" on page 570.	Yes	No
`BasicConstraintsExt` See "BasicConstraintsExt Rule" on page 574.	Yes	No
`UniqueSubjectName` See "UniqueSubjectNameConstraints Rule" on page 542.	Yes	No

Table 19.1 Default policy rules created for a Certificate Manager and Registration Manager (Continued)

Policy rule name	Created for Certificate Manager	Created for Registration Manager
`SubjectAltNameExt` See "SubjectAltNameExt Rule" on page 673.	Yes	Yes
`GenericASN1Ext` See "GenericASN1Ext Rule" on page 612.	Yes	Yes
`NameConstraintsExt` See "NameConstraintsExt Rule" on page 641.	Yes	No
`PolicyConstraintsExt` See "PolicyConstraintsExt Rule" on page 661.	Yes	No
`IssuerRule` See "IssuerRule Rule" on page 511.	Yes	No
`PolicyMappingsExt` See "PolicyMappingsExt Rule" on page 665.	Yes	No
`SubCANameCheck` See "SubCANameConstraints Module" on page 537.	Yes	No
`SigningAlgRule` See "SigningAlgRule Rule" on page 536.	Yes	No

To modify a policy rule in the CMS configuration:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select the subsystem to which the policy rule you want to modify belongs.
4. Select Policies.

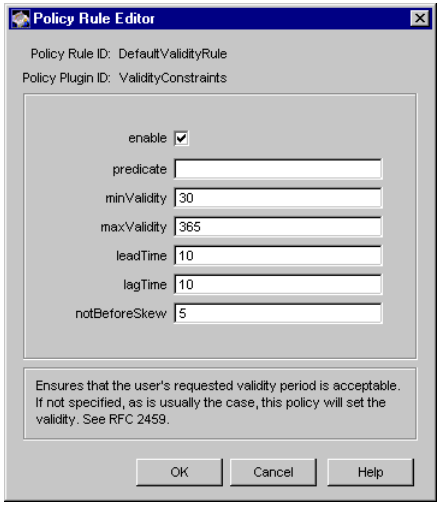
   The Policy Rules Management tab appears (Figure 19.2). It lists configured policy rules. The default rules are listed in Table 19.1.

5. In the Policy Rule list, select a rule that you want to modify.

   For the purposes of this instruction, assume that you selected the rule named `DefaultValidityRule`.

6. Click Edit/View.

   The Policy Rule Editor window appears, showing how this rule is configured. An example is shown below.



7. Make the necessary changes and click OK.

   You are returned to the Policy Rules Management tab.

8. Repeat steps 5 through 7 for the remaining rules.

9. Click Refresh.

Step 3. Delete Unwanted Policy Rules

You can delete any unwanted policy rules from the CMS configuration. If you think you might need a rule in the future, instead of deleting it from the configuration you should disable it by unchecking the `enable` parameter. In this way, you can avoid re-creating the rule in the future. Because the

subsystems subject end-entity requests only to rules that are currently enabled (see "Policy Processor" on page 497), keeping unwanted rules in the disabled state in the configuration does not affect policy decisions made by a subsystem.

To delete a policy rule from the CMS configuration:

1. In the Policy Rules Management tab, select the rule you want to delete and click Delete.

2. When prompted, confirm the delete action.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. Don't restart the server yet; you can do so after you've made all the required changes.

Step 4. Add New Policy Rules

Adding a policy rule to the CMS configuration involves creating a new instance of an already registered policy plug-in module, assigning a unique name for the instance, and entering appropriate values for the parameters that define the module you want to create an instance of.

When you add a policy rule, the CMS configuration gets updated with policy-specific information. Keep the following points in mind:

- When naming a policy instance (or rule), be sure to formulate the name using any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-); other characters and spaces are not allowed. For example, you can type `My_Policy_Rule` or `MyPolicyRule` as the instance name, but not `My Policy Rule`.

- The status of the rule, enabled or disabled, depends on whether you check or uncheck the `enable` parameter. A subsystem subjects certificate requests only to rules that are enabled.

- The server does not automatically reorder rules. Be sure to change the order of the rule, if required. For information on reordering rules, see "Step 5. Reorder Policy Rules" on page 705.

Figure 19.3 shows the policy modules registered with a Certificate Manager. The Registration Manager also has a similar list. Table 19.2 summarizes the default modules registered with both Certificate Manager and Registration Manager. If you have registered any custom policy modules (see "Registering a Policy Module" on page 709), they too will be available for selection.

Figure 19.3 Default policy modules registered with a Certificate Manager

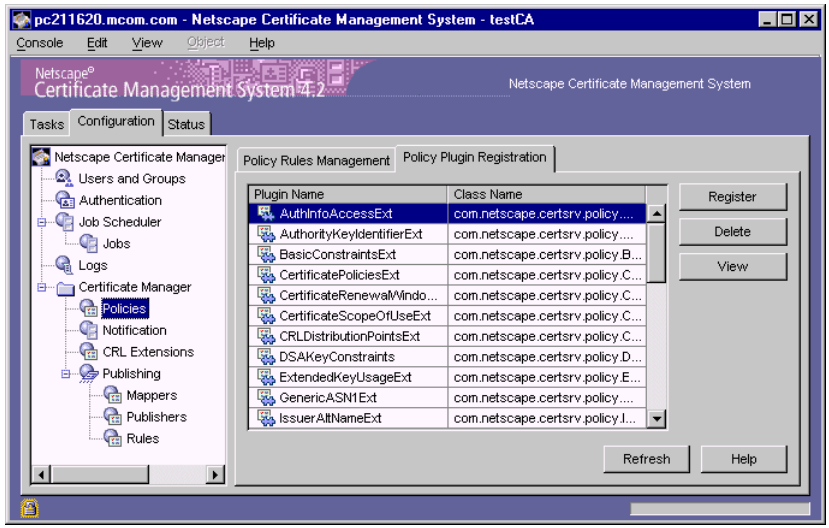


Table 19.2 Policy modules registered with a Certificate Manager and Registration Manager

Policy plug-in module name and description	Provided with Certificate Manager	Provided with Registration Manager
`AuthInfoAccessExt` See "Authority Information Access Extension Policy" on page 558.	Yes	Yes
`AuthorityKeyIdentifierExt` See "Authority Key Identifier Extension Policy" on page 566.	Yes	No
`BasicConstraintsExt` See "Basic Constraints Extension Policy" on page 570.	Yes	No

Table 19.2 Policy modules registered with a Certificate Manager and Registration Manager (Continued)

Policy plug-in module name and description	Provided with Certificate Manager	Provided with Registration Manager
`CertificatePoliciesExt` See "Certificate Policies Extension Policy" on page 574.	Yes	Yes
`CertificateRenewalWindowExt` See "Certificate Renewal Window Extension Policy" on page 580.	Yes	Yes
`CertificateScopeOfUseExt` See "Certificate Scope of Use Extension Policy" on page 586.	Yes	Yes
`CRLDistributionPointsExt` See "CRL Distribution Points Extension Policy" on page 591.	Yes	Yes
`DSAKeyConstraints` See "DSA Key Constraints Policy" on page 505.	Yes	Yes
`ExtendedKeyUsageExt` See "Extended Key Usage Extension Policy" on page 598.	Yes	Yes
`GenericASN1Ext` See "Generic ASN.1 Extension Policy" on page 605.	Yes	Yes
`KeyAlgorithmConstraints` See "Key Algorithm Constraints Policy" on page 512.	Yes	Yes
`IssuerAltNameExt` See "Issuer Alternative Name Extension Policy" on page 612.	Yes	Yes
`IssuerConstraints` See "Issuer Constraints Policy" on page 509.	Yes	No
`KeyUsageExt` See "Key Usage Extension Policy" on page 618.	Yes	Yes
`NameConstraintsExt` See "Name Constraints Extension Policy" on page 632.	Yes	No

Table 19.2 Policy modules registered with a Certificate Manager and Registration Manager (Continued)

Policy plug-in module name and description	Provided with Certificate Manager	Provided with Registration Manager
`NSCComment` See "Netscape Certificate Comment Extension Policy" on page 642.	Yes	Yes
`NSCertTypeExt` See "Netscape Certificate Type Extension Policy" on page 647.	Yes	Yes
`OCSPNoCheckExt` See "OCSP No Check Extension Policy" on page 653.	Yes	Yes
`PinPresent` See "PIN Present Policy" on page 514.	Yes	Yes
`PolicyConstraintExt` See "Policy Constraints Extension Policy" on page 657.	Yes	No
`PolicyMappingsExt` See "Policy Mappings Extension Policy" on page 661.	Yes	No
`PrivateKeyUsagePeriodExt` See "Private Key Usage Period Extension Policy" on page 666.	Yes	Yes
`RenewalConstraints` See "Renewal Constraints Policy" on page 518.	Yes	Yes
`RenewalValidityConstraints` See "Renewal Validity Constraints Policy" on page 525.	Yes	Yes
`RevocationConstraints` See "Revocation Constraints Policy" on page 522.	Yes	Yes
`RSAKeyConstraints` See "RSA Key Constraints Policy" on page 529.	Yes	Yes
`SigningAlgorithmConstraints` See "Signing Algorithm Constraints Policy" on page 533.	Yes	Yes

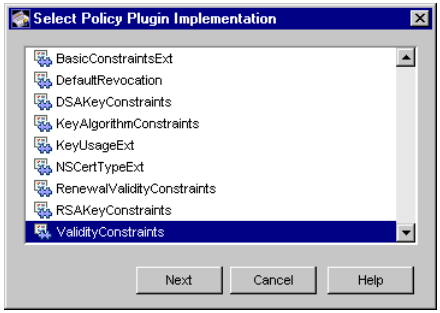
Table 19.2 Policy modules registered with a Certificate Manager and Registration Manager (Continued)

Policy plug-in module name and description	Provided with Certificate Manager	Provided with Registration Manager
`SubjectAltNameExt` See "Subject Alternative Name Extension Policy" on page 668.	Yes	Yes
`SubCANameCheck` See "Subordinate CA Name Constraints Policy" on page 536.	Yes	No
`SubjectDirectoryAttributesExt` See "Subject Directory Attributes Extension Policy" on page 675.	Yes	Yes
`SubjectKeyIdentifierExt` See "Subject Key Identifier Extension Policy" on page 679.	Yes	Yes
`UniqueSubjectName` See "Unique Subject Name Constraints Policy" on page 539.	Yes	No
`ValidityConstraints` See "Validity Constraints Policy" on page 543.	Yes	Yes

To add a new policy rule to the CMS configuration:

1. In the Policy Rules Management tab, click Add.

   The Select Policy Plugin Implementation window appears. It lists registered policy plug-in modules. The default modules are listed in Table 19.2.

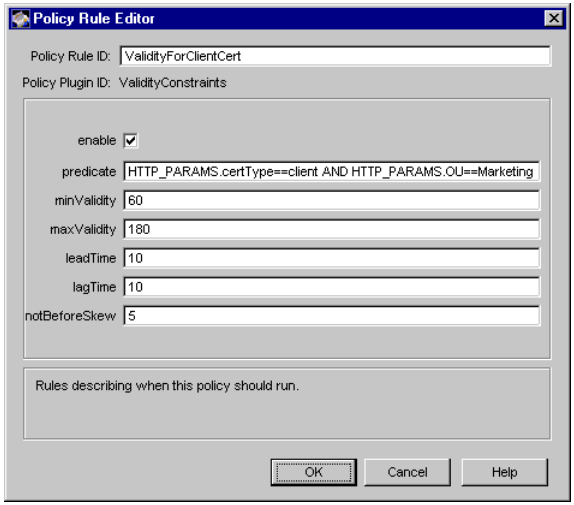


2. Select a plug-in module.

   For the purposes of this instruction, assume that you selected the `ValidityConstraints` module.

3. Click Next.

   The Policy Rule Editor window appears. It lists the configuration information required for this policy rule.



4. Enter the appropriate information.

   Policy Rule ID. Type a unique name that will help you identify the rule. Be sure to formulate the name using any combination of letters (aA to zZ), digits (0 to 9), an underscore (_), and a hyphen (-). For example, you can type `My_Policy_Rule` or `MyPolicyRule` as the instance name, but not `My Policy Rule`.

   enable. Check the box to enable the rule (default). If you enable the rule and set the remaining parameters correctly, the server sets the configured validity period in certificates specified by the `predicate` parameter.

   Uncheck the box to disable the rule. If you disable the rule, the server does not set the configured validity period in certificates; it sets the validity period to the one specified in the request.

predicate. Type the predicate expression for this rule. If you want this rule to be applied to all certificate requests, leave the field blank (default). To form a predicate expression, see "Using Predicates in Policy Rules" on page 490.

minValidity. Type the minimum validity period, in days, for certificates. The value must be an integer greater than zero and less than the value you will type for the `maxValidity` parameter next. The default value is 180 days.

maxValidity. Type the maximum validity period, in days, for certificates. The value must be an integer greater than zero and also greater than the value you typed for the `minValidity` parameter. The default value is 730 days.

leadTime. Type the lead time, in minutes, for certificates. For a certificate renewal request to pass the renewal validity constraints policy, the value of the `notBefore` attribute in the certificate request must not be more than value of the `leadTime` parameter in the future, relative to the time when the policy rule is run. The default value is 10 minutes.

The `notBefore` attribute value specifies the date on which the certificate validity begins.

lagTime. Type the lag time, in minutes, for certificates. For a certificate renewal request to pass the renewal validity constraints policy, the value of the `notBefore` attribute in the certificate request must not be more than the value of the `lagTime` in the past, relative to the time when the policy is run. The default value is 10 minutes.

The `notBefore` attribute value specifies the date on which the certificate validity ends.

notBeforeSkew. Type the number of minutes to subtract from the current time when creating the value for the certificate's `notBefore` attribute. It can help some clients with incorrectly set clocks use the new certificate after downloading. For example, if the certificate is issued at 11:30 a.m. and the clock settings of the client into which the certificate is downloaded is 11:20 a.m., the certificate cannot be used for 10 minutes. Setting the value of the `notBeforeSkew` parameter to 10 minutes would adjust the value of the notBefore parameter to 11:20 a.m.—thus making the certificate usable following the down load. The default value is 5 minutes.

5. Click OK.

   You are returned to the Policy Rules Management tab.

6. Repeat steps 1 through 5 and create additional rules, if required.

Step 5. Reorder Policy Rules

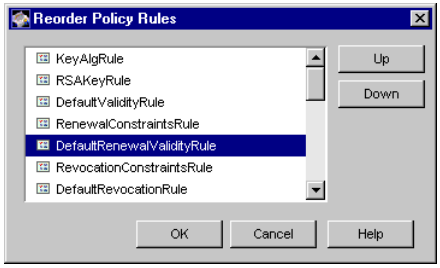
For maintaining priority levels, Certificate Management System supports a linear list of policy rules in increasing order of priority. This means that for a given policy category in the configuration file, a policy configuration with a lower priority precedes one with a higher priority. This simple linear listing avoids the need to have explicit locking on request attributes to prevent conflicting changes. By ordering the rules, you introduce a concurrency control whereby a higher-priority rule configuration overwrites any changes made by a lower-priority rule configuration that precedes it.

You may want to specify policies at different priority levels for the same operation depending on the end-entity information. For example, authentication policies, if any, need to precede others in the list.

To reorder policy rules in the CMS configuration:

1. In the Policy Rules Management tab, click Reorder.

   The Reorder Policy Rules window appears. It lists configured policy rules in the order in which they are executed by the subsystem. Keep in mind that the server executes the rules on a first-come-first-served basis, overwriting the configuration determined by the previous rule, if any.



2. To change the order of a rule, select it in the list and click the Up or Down button, as appropriate.

3. When you have the correct order, click OK.

   You are returned to the Policy Rules Management tab.

4. To view the updated configuration, click Refresh.

Step 6. Restart the Server

This step is required only if you were prompted to restart the server in any of the previous steps.

To restart the server from the CMS window:

1. Click the Tasks tab
2. Click Restart the Server.

Step 7. Test Policy Configuration

To make sure that you've configured the server correctly, request a certificate and check the certificate for details such as for validity period, key type and size, and extensions.

- Step A. Enroll for a Certificate
- Step B. Approve the Request
- Step C. Check the Certificate Details

Step A. Enroll for a Certificate

The steps outlined below explain how to request a personal certificate from the Certificate Manager using the manual enrollment method (see "Manual Authentication" on page 323). If you've configured the Certificate Manager for automated certificate issuance, for example the directory-based enrollment, you can use the appropriate form and request a certificate.

To request a client or personal certificate from the Certificate Manager:

1. Open a web browser window.
2. Go to the End Entity Services interface of the Certificate Manager you configured (or the Registration Manager that's connected to this Certificate Manager). The URL is in this form:

   `https://<host_name>:<end_entity_HTTPS_port>` or
   `http://<host_name>:<end_entity_HTTP_port>`
3. In the left frame, under Browser, click Manual.

   This opens the manual enrollment form.
4. Fill in all the values and submit the request.

   The client prompts you to enter the password for your key database.
5. When you enter the correct password, the client generates the key pairs.

   Do not interrupt the key-generation process.

Step B. Approve the Request

This step is required if you used the manual enrollment form for requesting the certificate. The request you submitted is waiting in the agent queue for approval by an agent.

To approve the request:

1. Go to the Certificate Manager's Agent Services interface.

   The URL is in this format: `https://<host_name>:<agent_port>`
2. In the left frame, click the link that says List Requests.
3. In the form that appears, select the "Show pending requests" option and click Find.

   You should see your request in the list of pending requests.
4. Locate the request you submitted and approve the request.

   You should see a confirmation page indicating that the certificate has been issued. Don't close the page until you finish the next step.

Step C. Check the Certificate Details

Verify that the certificate contains the required details. Be sure to check the Extension section to see if it contains all the required extensions.

Managing Policy Plug-in Modules

This section explains how to use the CMS window to perform the following operations:

- Registering a Policy Module
- Deleting a Policy Module

For information on adding or changing policy-specific information in the configuration file, see "Policy Parameters in the Configuration File" on page 689.

Registering a Policy Module

You can register new policy plug-in modules in a subsystem's policy framework. Registering a new policy module involves specifying the name of the module and the full name of the Java class that implements the policy interface. For example, you can add a policy implementation, named as follows, to the Data Recovery Manager's policy framework:

```
com.netscape.policy.KeyArchivalPolicy
```

Before registering a plug-in module, be sure to put the Java class for the module in the `classes` directory (the implementation must be on the class path).

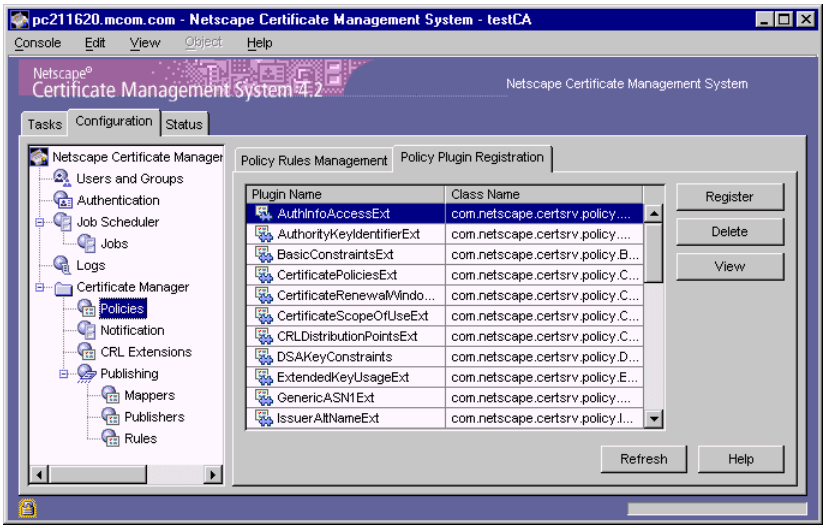
To register a policy module in a subsystem's policy framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.

3. In the navigation tree, select the subsystem that will use the module you want to register.

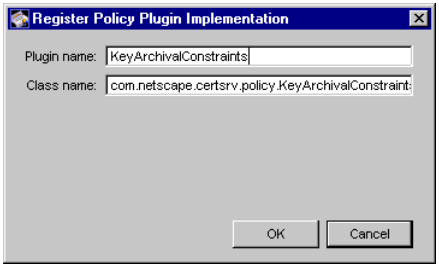
4. Select Policies, and then in the right pane, select the Policy Plugin Registration tab.

   The Policy Plugin Registration tab appears. It lists registered policy plug-in modules.



5. Click Register.

   The Register Policy Plugin Implementation window appears.



6. Specify information as appropriate:

   Plugin name. Type a name for the plug-in module.

Class name. Type the full name of the class for this module—that is, the path to the implementing Java class. If this class is part of a package, be sure to include the package name. For example, if you are registering a class named `myPolicy` and if this class is in a package named `com.myCompany`, type `com.myCompany.myPolicy`.

7. Click OK.

   You are returned to the Policy Plugin Registration tab.

8. To view the updated configuration, click Refresh.

Deleting a Policy Module

You can delete unwanted policy plug-in modules using the CMS window. Before deleting a module, be sure to delete all the policy rules that are based on this module; see "Step 3. Delete Unwanted Policy Rules" on page 697.

To delete a policy module from a subsystem's policy framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).

2. Select the Configuration tab.

3. In the navigation tree, select the subsystem that registers the module you want to delete.

4. Select Policies, and then in the right pane, select the Policy Plugin Registration tab.

   The Policy Plugin Registration tab appears. It lists registered policy modules. For information about this tab, see "Policy Plugin Registration Tab" on page 688.

5. In the Plugin Name list, select the module you want to delete and click Delete.

6. When prompted, confirm the delete action.

7

Publishing

Chapter

20

Introduction to Publishing Certificates and CRLs

In Certificate Management System, publishing refers to the ability of the Certificate Manager to publish certificates, CRLs, and other certificate-related objects to any of the supported repositories—an LDAP-compliant directory, a flat file, and an online validation authority—using the appropriate protocol. Configuring the Certificate Manager for publishing is optional—you can turn this feature off without affecting any of the certificate issuance and management operations handled by the server.

This chapter explains how a Certificate Manager works with the above mentioned repositories. The chapter has the following sections:

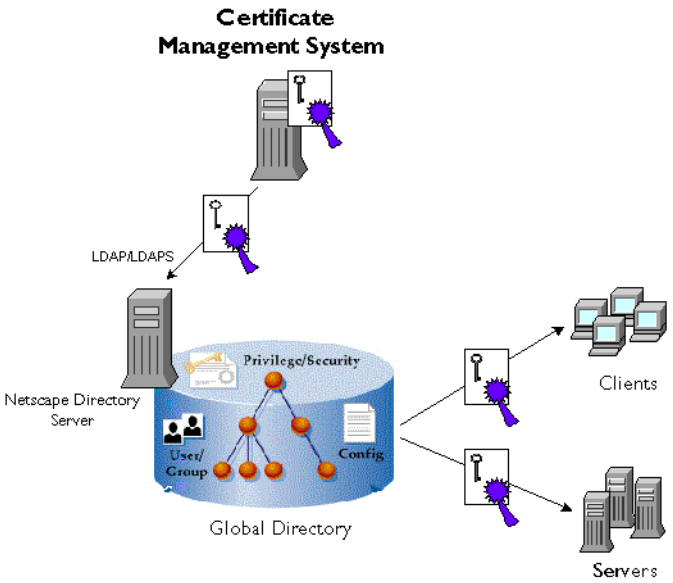
Publishing of Certificates

The Certificate Manager can publish certificates (ones that it issues) to an LDAP-compliant directory and a flat file. Sections that follow explain each of these in detail.

Publishing of Certificates to a Directory

Large corporations typically use Lightweight Directory Access Protocol (LDAP) directories, such as Netscape Directory Server, to store and manage corporatewide data, including user and group information and network resource data. If you have deployed an LDAP-compliant directory, you can configure the Certificate Manager to automatically publish your CA and end-entity certificate-related information to that directory. For example, if you have configured the Certificate Management System to employ directory-based authentication, you should consider publishing the CA and end-entity certificates to the same directory. This way, you can keep your users' security credentials with the rest of the user information (see Figure 20.1).

Figure 20.1 Publishing certificates to a directory for distribution



The ability of a Certificate Manager to publish certificates, CRLs, and other certificate-related objects to a directory using the LDAP or LDAPS protocol is called *LDAP publishing* and the directory to which it publishes is called the *publishing directory*. Note that configuring the Certificate Manager for LDAP publishing is optional—you can turn this feature off without affecting any of the certificate issuance, renewal, and revocation operations handled by the server.

You can configure the Certificate Manager to automatically publish certificates to the directory every time a certificate is issued and at a predetermined interval—for example, every day or once every week. Privileged users (administrators and agents) can also manually initiate the LDAP publishing process.

Figure 20.2 illustrates LDAP publishing by the Certificate Manager when a certificate requested via the manual-enrollment process is issued.

Figure 20.2 Publishing by a Certificate Manager

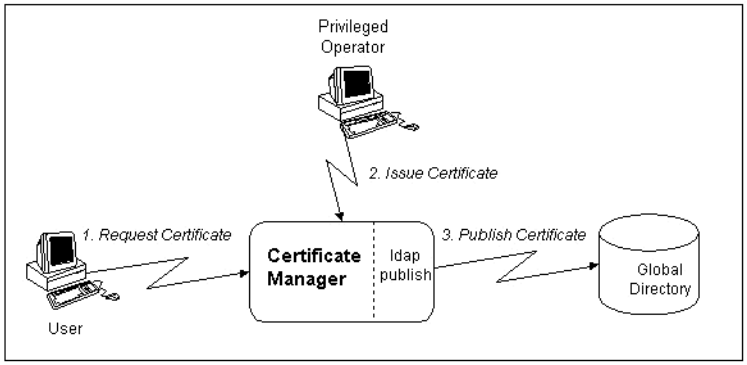
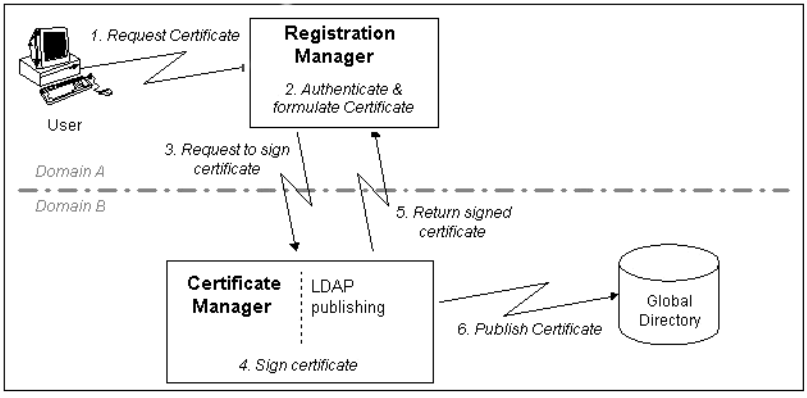


Figure 20.3 illustrates how certificates requested via a Registration Manager get published to the directory.

Figure 20.3 Publishing of certificates requested via a Registration Manager



Timing of Directory Updates

If the LDAP directory is properly configured to work with the Certificate Manager (and vice versa), any changes to the certificate information in the Certificate Manager are automatically made also in the publishing directory.

The publishing directory is updated at these times:

- When the Certificate Manager starts up, it publishes its *CA signing certificate* to the directory.
- When the Certificate Manager issues a new certificate, it stores a copy of the certificate in its internal database and then publishes the certificate to the directory.
- When the Certificate Manager revokes a certificate, it marks the copy of the certificate in its internal database as revoked and then unpublishes or removes the revoked certificate from the directory.

Table 20.1 summarizes the above-listed actions of the Certificate Manager. The table also indicates how the Certificate Manager populates an LDAP directory, if configured for publishing. Note that certificates (and CRLs) are published as DER-encoded binary blobs.

Table 20.1 Details of objects published by the Certificate Manager

Object	Action and Timing	LDAP entry	LDAP attribute
End-entity certificate	Publishing occurs when a certificate is issued or renewed	End-entity's entry	`userCertificate;binary`
	Unpublishing (removal) occurs when a certificate is revoked or expired	End-entity's entry	`userCertificate;binary`
CA certificate	Publishing occurs when the Certificate Manager is started	CA's entry	`caCertificate;binary`
CRL (full)	Publishing (replacement) occurs when a new CRL is generated	CA's entry	`certificateRevocation List;binary`

The Certificate Manager cannot update the directory in the following cases:

- If an end-entity entry is not present or if an entry cannot be found to publish the certificate.
- If the directory's schema doesn't include the appropriate attributes. To configure the directory for LDAP publishing, see "Step 2. Set Up the Directory for Publishing" on page 789. Note that the Certificate Manager publishes to the `userCertificate;binary` attribute, which is an LDAP v3 standard. Unless you are using a non-standards compliant directory, this situation shouldn't arise.
- When the directory is unreachable because maintenance work is being performed, or because of network or system failures.

Note that the Certificate Manager's LDAP publishing action happens as a separate transaction from any certificate operation (such as issuance); the operation of a certificate is not affected by whether it was successfully published or not.

Directory Update Process

As indicated in Table 20.1 on page 718, when a Certificate Manager is requested to issue a certificate, update certificate information, or publish a CRL, it automatically updates the corresponding entry in the configured directory with relevant information. To locate the correct directory entry, the Certificate Manager relies on object-mapping rules, which can be defined using the mapper modules; for details, see "Mapper Modules" on page 732. Once an entry is located, to publish the object to the correct attribute, the server relies on object-publishing rules, which can be defined with the help of publisher modules; for details, "Publisher Modules" on page 749.

Similarly, when you revoke a certificate, the Certificate Manager uses the object mapping and publishing rule to locate and delete the corresponding certificate from the directory.

For instruction on how to configure the Certificate Manager to publish to a directory, see "Publishing Certificates and CRLs to a Directory" on page 786.

Directory Synchronization

The Certificate Manager and the publishing directory can become out of sync if certificates are issued or revoked while Directory Server is down. Certificates that were issued or revoked need to be published or unpublished manually when Directory Server comes back up.

To help find certificates that are out of sync with the directory—that is, valid certificates that are not in the directory and revoked or expired certificates that are still in the directory—the Certificate Manager keeps a record of whether a certificate in its internal database has been published to the directory. If the Certificate Manager and the publishing directory become out of sync, you can use the Update Directory option in the Certificate Manager Agent Services interface to synchronize the publishing directory with the internal database.

The following choices are available for synchronizing the directory with the internal database:

- Search the internal database for certificates that are out of sync and publish or unpublish accordingly.
- Publish certificates that were issued from time A to time B while Directory Server was down. Similarly, unpublish certificates that were revoked or that expired while Directory Server was down.
- Publish or unpublish a range of certificates based on serial numbers (from serial number *xx* to serial number *yy*).

For instructions, see "Manually Updating Certificates in the Directory" on page 838.

Publishing of Certificates to a Flat File

In addition to publishing to a directory, the Certificate Manager can publish the certificate to a file. The certificate is published as a DER-encoded binary blob and applications that are capable of reading such data may read the file for certificate information. For example, you can customize the sample code for Flat File CRL and certificate publisher can be customized to store certificates and CRLs in a relational database management system.

For instructions on how to configure a Certificate Manager to publish certificates to a flat file, see "Publishing Certificates and CRLs to Flat Files" on page 840.

Publishing of CRLs

Server and client applications that use public-key certificates as tokens of identification need access to information about the validity of a certificate; because one of the factors that determines the validity of a certificate is its revocation status, these applications need to know whether the certificate being validated has been revoked. In that regard, the CA has a responsibility to do the following:

- Revoke the certificate if any of the certificate assertions becomes false—for example, if the subject's key gets compromised or the status of the subject's role or right changes. (See "Reasons for Revoking a Certificate" on page 722.)
- Make the revoked certificate available to parties or applications that need to verify its validity status.

One of the standard methods for conveying the revocation status of certificates is by publishing a list of revoked certificates. This list is known as the *certificate revocation list* (CRL). The CRL is a publicly available list of certificates that have been revoked.

A CRL is issued and digitally signed by the certificate authority (CA) that issued the certificates listed in the CRL. The CA's function includes creating the CRLs periodically and distributing them to other applications. For example, the CA may publish the CRL to a global directory which other applications may use for checking the revocation status of a certificate or from which other applications can retrieve the CRL.

In Certificate Management System, the Certificate Manager creates the CRL and publishes it to the configured repositories; this is explained in "Supported Methods for Verifying Revocation Status of Certificates" on page 724. Configuring a Certificate Manager to publish CRLs is optional. Note that the Registration Manager cannot create or publish the CRL.

Normally, whenever a certificate is revoked (by administrators, agents, or end users), the Certificate Manager automatically updates the status of the certificate in its internal database—it marks the copy of the certificate in its internal database as revoked and removes the revoked certificate from the directory, if the Certificate Manager is configured to do so. In addition to certificates, the Certificate Manager also maintains a CRL in its internal database. You can configure the Certificate Manager to generate the CRL every time a certificate is revoked and at periodic intervals.

You can also configure the Certificate Manager to generate and publish CRLs conforming to X.509 (either version 1 or version 2) standards by enabling or disabling the CRL extension-specific modules in the server's configuration. Note that the server supports standard CRL extensions that are explained in "CRL Extension Modules" on page 763.

For instructions on how to configure a Certificate Manager to publish CRLs, see "Configuring a Certificate Manager for Publishing" on page 785.

Reasons for Revoking a Certificate

A Certificate Manager can revoke any certificate it has issued. A certificate needs to be revoked if one or more of the following situations occur:

- The owner of the certificate has changed status and no longer has the right to use the certificate.
- The private key of a certificate owner has been compromised.
- The certificate owner doesn't want to use the certificate.
- The private key of the CA that issued the certificate has been compromised.

A certificate can be revoked by administrators, agents, and end entities, such as end users and individual server administrators. Agents and administrators (with agent privileges) can revoke certificates by using the forms provided in the agent interface. Administrators, agents, and end users can revoke certificates by using the forms provided in the Revocation tab of the end-entity interface. Note that end users can revoke only their own certificates, whereas agents and administrators can revoke any certificates issued by the server. End users are required to authenticate to the server in order to revoke their certificate; see "Authentication of End Users During Certificate Revocation" on page 314.

Whenever a certificate is revoked, the Certificate Manager updates the status of the certificate in its internal database. This way, the server keeps track of all revoked certificates in its internal database and it makes the revoked list of certificates public (by publishing it to a central repository) to notify other users that the certificates in the list are no longer valid.

Revocation Checking by Netscape Clients

At the time of this writing, Netscape Communicator versions 4.7 and later, when used in conjunction with the security module called Netscape Personal Security Manager, enable automatic revocation-status verification of certificates using the OCSP protocol. The section "Publishing of CRLs to an Online Validation Authority" on page 726 explains how the revocation status of a certificate is verified in an OCSP-compliant PKI setup.

Earlier versions of Netscape client products do not have the ability to automatically check to see whether a certificate has been revoked. However, these clients do give the user the ability to check the revocation status of a certificate if it includes the `NetscapeRevocationURL` extension. For details about this extension, check this site:

```
http://home.netscape.com/eng/security/cert-exts.html
```

In addition, from the Retrieval tab of the end-entity interface, Netscape client users can manually check the revocation status of a particular certificate and automatically import the latest version of the CRL into their browsers. If your users are not using Netscape clients, they can download the latest CRL in binary form to a local file, and then import this file into their browsers by an appropriate method. Users can also view the header information of the master or full CRL published by the Certificate Manager, which contains the date and time of the latest update, and then compare this information to that in their browser's CRL to see if they have the latest version.

Revocation Checking by Netscape Servers

Because Netscape servers currently cannot check the revocation status of a certificate, you should use other forms of access control. For example, you can remove individual users from access groups to prevent them from accessing the server.

Because Certificate Management System can check the revocation status of the certificates that it issues, you do not need to rely on other forms of access control.

Supported Methods for Verifying Revocation Status of Certificates

Revocation status of a certificate can be made available to PKI entities by publishing the CRL to various repositories. To aid you in this process, the Certificate Manager supports publishing of CRLs to the following repositories:

- An LDAP-compliant directory
- A flat file
- An Online Certificate Status Protocol (OCSP)-compliant validation authority or OCSP responder

Applications in your enterprise may use any of these repositories to verify the revocation status of a certificate.

Publishing of CRLs to an LDAP Directory

The Certificate Manager can publish the CRL to an LDAP-compliant directory using the LDAP protocol or LDAP over SSL (LDAPS) protocol, and applications can retrieve the CRL over HTTP. Support for retrieving CRLs over HTTP enables some browsers, such as Netscape Communicator, to automatically import the latest CRL from the directory that receives regular updates from the Certificate Manager. The browser can then use the CRL to automatically check all certificates to ensure that they have not been revoked.

For applications that are incapable of retrieving the CRL over HTTP, the Certificate Manager also supports retrieval of the CRL in binary form. For example, if the browser you've deployed doesn't support CRL retrieval over HTTP, your users may download the CRL to a local file and then import the file into their browsers by an appropriate method.

You can configure a Certificate Manager to publish the CRL it maintains to a directory, for example, to the same directory in which end-entity certificates are published. If you configure the Certificate Manager and directory to work

properly, any changes to the CRL information in the Certificate Manager are automatically updated in the publishing directory. Note that the server publishes the CRL to the `certificateRevocationList;binary` attribute of the CA's entry in the directory. To locate the correct directory entry, the Certificate Manager uses object mapping rules; to publish the CRL to the correct attribute of the located entry, the server uses publishing rules. For details, see "Mapper Modules" on page 732 and "Publisher Modules" on page 749.

Directory updates take place depending on how you configure the Certificate Manager—that is, publish the CRL to the directory every time a certificate is revoked or at specific intervals, or both. It's important to understand that when the Certificate Manager revokes a certificate, it marks the copy of the certificate in its internal database as revoked and generates CRL before publishing it to the configured directory. For example, if you configure the server to publish the CRL every time a certificate is revoked, CRL will be generated whenever a certificate is revoked.

For instructions on configuring a Certificate Manager for publishing CRLs to a directory, see "Publishing Certificates and CRLs to a Directory" on page 786.

If the Certificate Manager and publishing directory become out of sync for some reason, privileged users (administrators and agents) can also manually initiate the publishing process. For instructions, see "Manually Updating the CRL in the Directory" on page 839.

CRL Issuing Points

Because CRLs can grow very large, several methods have been developed to minimize overhead of retrieving and delivering large CRLs. One of these methods is based on partitioning the entire certificate space and associating a separate CRL with every partition. This partition is called a *CRL issuing* or *distribution point*—it is the location where a subset of all the revoked certificates are maintained. Partitioning can be based on revocation reason, on whether the revoked certificate is a CA certificate or end-entity certificate, on end users' names, and so on. Each issuing point is identified by a set of names, which can be in various forms.

Once the issuing points have been defined, they can be included in certificates so that an application that needs to check the revocation status of a certificate can access the CRL issuing points specified in the certificate instead of the master or main CRL—the application would check the CRL maintained at the issuing point, which would be smaller in size compared to the master CRL, and thus speed up the revocation-status-checking process.

CRL distribution points can be associated with certificates using the `CRLDistributionPoint` extension in the certificates.

By default, the Certificate Manager only generates and publishes a single CRL, identified as the *master CRL*. However, for interoperatability purposes, the server does enable you to add the `CRLDistributionPoint` extension to the certificates it issues. For details, see "CRL Distribution Points Extension Policy" on page 591.

Publishing of CRLs to Flat Files

In this method, the Certificate Manager publishes the CRL to a file. The CRL is published as a DER-encoded binary blob and applications that are capable of reading such data may read the file for CRL information. You may also use the file to import the CRL to other repositories.

For instructions on how to configure a Certificate Manager to publish CRLs to a flat file, see "Publishing Certificates and CRLs to Flat Files" on page 840.

Publishing of CRLs to an Online Validation Authority

Certificate Management System supports the Online Certificate Status Protocol (OCSP) as defined in the PKIX standard RFC 2560 (see `http://www.ietf.org/rfc/rfc2560.txt`). The Certificate Manager can publish a CRL to an OCSP-compliant online validation authority or server, such as ValiCert Certificate VA™.

The OCSP protocol enables OCSP-compliant applications to determine the state of a certificate, including the revocation status, without having to directly check a CRL published by a CA to the validation authority. The validation authority, which is also called an OCSP responder, does the checking for the application.

An OCSP-compliant PKI setup generally includes the following, which work together to verify the revocation status of a certificate:

- A CA, which issues and revokes certificates, and periodically publishes a CRL to the OCSP responder.
- An OCSP responder, which maintains the CRL it receives periodically from the CA and, when queried by an OCSP-compliant client about the status of a certificate, sends a digitally signed response.

- OCSP-compliant applications, which, when trying to validate a certificate, query the appropriate OCSP responder (using the OCSP protocol) for the status of the certificate. The applications determine the location of the OCSP responder by using the Authority Information Access extension in the certificate being validated. (Certificate Management System allows you to add this extension to certificates; see "Authority Information Access Extension Policy" on page 558.)

The revocation-status verification process has two parts:

1. When a certificate's status needs to be verified, the OCSP client (an OCSP-compliant application) sends a request to the OCSP responder for verification and waits for a response from the responder.

   The OCSP request that the client submits generally contains all the information required by the responder to identify the certificate whose status it needs to determine.

   (Consider this process is similar to a cashier scanning your credit card and waiting for a response from the credit-card processing unit. The scanning unit sends identifying information, such as the credit card number, its type, validity period, and so on.)

2. Upon receipt of the request, the OCSP responder determines if the request contains all the information required by the responder to process it.

   - If the request lacks any information required by the responder to process it or if the responder is not configured to provide the requested service to the client, the responder sends a rejection notification to the client. The responder also writes an appropriate error message to its log file.

   - If the request meets all the criteria, the responder returns a response to the client that requested it: it checks its list of revoked certificates for the one whose status is being requested, verifies its status, composes a report, signs the report, and sends the report to the client.

Note that every response that the client receives, including a rejection notification, is digitally signed by the responder; the client is expected to verify the signature to ensure that the response came from the responder to which it submitted the request. The key the responder uses to sign the message

depends on how the OCSP responder is deployed in a PKI setup. RFC 2560 recommends that the key used to sign the response belong to one of the following:

- The CA that issued the certificate and whose status is being verified by the responder.
- A responder whose public key, which corresponds to the private key it uses to sign responses, is trusted by the client. Such a responder is called a *trusted responder*.
- A responder that holds a specially marked certificate issued to it directly by the CA that revokes the certificates and publishes the CRL. Possession of this certificate by a responder indicates that the CA has authorized the responder to issue OCSP responses for certificates revoked by the CA. Such a responder is called a *CA-designated responder* or a *CA-authorized responder*.

  Certificate Management System allows you to request OCSP responder certificates. The end-entity interface of both Registration Manager and Certificate Manager includes a form that allows you to manually request a certificate for the OCSP responder. The default enrollment form includes all the attributes (for example, `HTTP_PARAMS.certType==ocspResponder`) that identify the certificate as an OCSP responder certificate. The required extensions, such as OCSP No Check and OCSP Signing, can be added to the certificate when the certificate request is subjected to policy checking; see "OCSP No Check Extension Policy" on page 653 and "OCSPSigningExt Rule" on page 603.

The OCSP response that the client receives indicates the current status of the certificate as determined by the OCSP responder. The response could be any of the following:

- Good—specifying a positive response to the status inquiry. At a minimum, this positive response indicates that the certificate has not been revoked, but it does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the responder regarding the status of the certificate such as positive statement about issuance, validity, etc.
- Revoked—specifying that the certificate has been revoked, either permanently or temporarily.

- Unknown—specifying that the OCSP responder doesn't know about the certificate whose status is being requested by the client.

Based on the status, the client decides whether to validate the certificate.

Local OCSP Support

As mentioned in the preceding section, in addition to a CA, you also need an OCSP responder and OCSP-compliant clients if you want to set up an OCSP-compliant PKI setup. To aid you in this process, Certificate Management System bundles the following components:

- ValiCert Certificate VA™ (or Certificate VA), version 3.0.1—this is an online validation authority or OCSP responder. You can use this server to set up a CA-designated OCSP responder.

- Netscape Personal Security Manager (or Personal Security Manager)—this is an OCSP-compliant security plug-in module for Netscape Communicator. The module, in addition to many other features, enables Communicator to check certificate validity in real time using the OCSP protocol: it enables the client to read the authority information access extension in a certificate, locate the OCSP responder specified by the extension, request the revocation status of the certificate from the OCSP responder, and use the response to validate the certificate. For a brief introduction to Personal Security Manager, see page 51.

When you install Certificate Management System, the files for the Certificate VA are placed at this location: `<server_root>/cva301`

Similarly, the files for the Personal Security Manager, version 1.1, are placed at this location: `<server_root>/psm11`

Following CMS installation, you may choose to install these components by running the appropriate `setup` programs. By installing the Certificate VA, you can deploy an internal OCSP responder and not outsource your CRL to a publicly-accessible machine. For instructions on how to set up a local OCSP responder, see "Publishing CRLs to Online Validation Authority" on page 857.

Chapter

21

Modules for Publishing Certificates and CRLs

When a Certificate Manager is requested to issue a certificate or to update certificate information, it can automatically update the corresponding entry in the configured LDAP directory or the flat file with relevant information. Similarly, when a certificate is revoked, the Certificate Manager can automatically update the configured LDAP directory, the online-validation authority, or the flat file with relevant CRL information. To locate the correct entry and update it with relevant information, the Certificate Manager relies on object-mapping and publishing rules.

To enable you to construct these rules, the Certificate Manager provides a set of mapper and publisher plug-in modules. These modules are implemented as Java classes and are registered with the CMS publishing framework.

This chapter explains the mapper and publisher modules that are installed with a Certificate Manager—it lists and briefly describes the modules and then explains each one in detail. Before reading this chapter, you should have read the previous chapter, "Introduction to Publishing Certificates and CRLs" on page 715.

The chapter has the following sections:

Mapper Modules

You can configure a Certificate Manager to publish certificates to a directory or flat file, and to publish CRLs to a directory, online validation authority, or flat file. If you configure the Certificate Manager to publish to a directory, whenever the server issues a certificate or updates a certificate or CRL, it needs to locate the entry in the directory in order to update it. For example, to find the correct directory entry to update, the Certificate Manager needs to present Directory Server with search criteria (so that it can initiate an LDAP search operation); the Certificate Manager considers the search successful only if Directory Server returns a single LDAP entry that exactly matches the search criteria.

The Certificate Manager uses object-mapping rules to find the directory entry that needs to be updated. When configuring a Certificate Manager for publishing certificates and CRLs, you define mapping rules that help the server to construct appropriate search criteria that find the entry that needs to be updated.

Mapper modules help you configure the Certificate Manager to use specific rules to map or locate a specific entry, such as a CA's entry or an end-entity's entry, in a specified directory; once the correct entry is located, the server publishes the certificate or CRL to the correct attribute in the entry using a publisher rule, as explained in "Publisher Modules" on page 749.

Overview of Mapper Modules

By default, the Certificate Manager provides a set of mapper plug-in modules for mapping the CA certificate, end-entity certificates, and CRLs to appropriate entries in an LDAP directory; because it's not required to map entries in a flat file and online validation authority, no mapper modules are provided for mapping objects in a flat file or an online validation authority. Plug-in modules are implemented as Java classes and are registered in the CMS publishing framework. The Mapper Plugin Registration tab of the CMS window (Figure 21.1) lists all the modules and the corresponding classes that are registered by default with a Certificate Manager.

Figure 21.1 Default mapper modules registered with a Certificate Manager

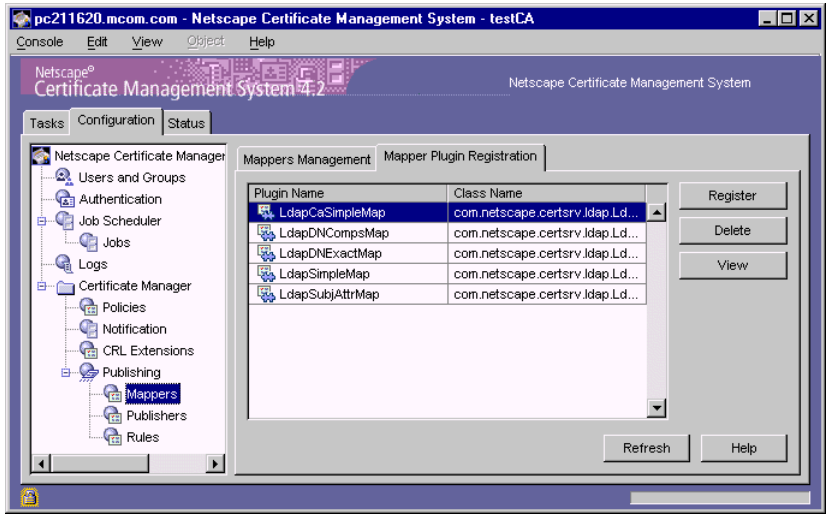


Table 21.1 lists the mapper modules provided for the Certificate Manager.

Table 21.1 Default mapper plug-in modules for mapping certificates and CRLs

Plug-in module name	Function
`LdapCaSimpleMap`	Maps the CA certificate to the CA's directory entry by formulating the entry's DN from components specified in the certificate's issuer name and attribute variable assertion (AVA) constants. Optionally, the plugin can also create an entry for the CA in the directory. For details, see "CA Certificate Mapper" on page 734.
`LdapDNCompsMap`	Maps a certificate to a directory entry by formulating the entry's DN from components (such as `CN`, `OU`, `O`, and `C`) in the certificate's subject name and using it as the search DN to locate the entry in the directory. For details, see "DN Components Mapper" on page 738.
`LdapDNExactMap`	Maps a certificate to a directory entry by searching for the entry whose DN exactly matches the certificate subject name. For details, see "Subject Name Mapper" on page 744.
`LdapSimpleMap`	Maps a certificate to a directory entry by formulating the entry's DN from components specified in the certificate's subject name and attribute variable assertion (AVA) constants. For details, see "Simple Mapper" on page 745.
`LdapSubjAttrMap`	Maps a certificate to a directory entry by searching for the entry that contains the LDAP attribute named `certSubjNameAttr` whose value exactly matches the certificate subject name. For details, see "Subject Attribute Mapper" on page 747.

After you take a look at the default mapper modules, if you determine that they do not meet your requirements entirely, you can develop a custom mapper module by implementing the following Java interface:

```
com.netscape.certsrv.ldappublish.ILdapMapper
```

For more information on this interface, check the CMS software development kit (SDK) installed at this location: `<server_root>/cms_sdk/sdkdocs`

Be sure to take a look at the samples available at this location: `<server_root>/cms_sdk/samples/publishing/mappers`

When developing a custom mapper module, you may want to intercept LDAP error 52 and reword it so that the correct error message gets logged. To give you an example, if the publishing directory has been stopped, the server logs the following message in its error and system logs:

```
Error publishing CRL MasterCRL: Cannot find a match in the LDAP
server for certificate. netscape.ldap.LDAPException: unable to
establish connection (52); DSA is unavailable.
```

Notice that the error message incorrectly says DSA is unavailable instead of Directory Server is unavailable.

For instructions on how to configure a Certificate Manager to use a mapper module, see "Step 3. Configure the Certificate Manager to Publish Certificates" on page 810.

CA Certificate Mapper

The `LdapCaSimpleMap` plug-in module implements the CA certificate mapper. This mapper enables you to configure a Certificate Manager to automatically create an entry for the CA in an LDAP directory and then map the CA's certificate to the directory entry by formulating the entry's DN from components specified in the certificate's subject name and attribute variable assertion (AVA) constants. For more information on AVAs, check the directory documentation.

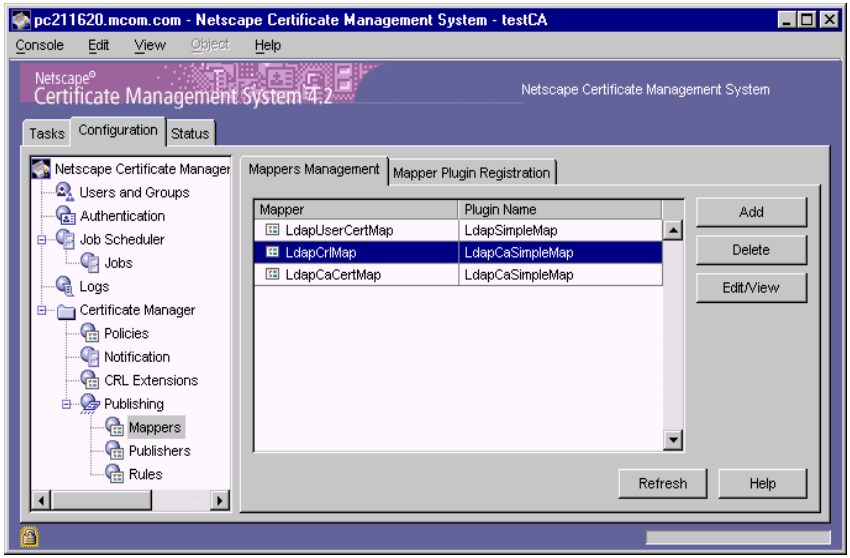
The CA certificate mapper allows you to specify whether to create an entry for the CA or to just map the certificate to an existing entry, or to do both; for example, you can choose to manually create an entry for the CA in the

directory and then configure the CA certificate mapper to just locate the entry by using attributes from the issuer name in the CA's signing certificate and AVA constants.

During installation, the Certificate Manager automatically creates two instances (called mappers) of the CA certificate mapper module (see Figure 21.2). The mappers are named as follows:

- `LdapCrlMap` for CRLs (see "LdapCrlMap Mapper" on page 738)
- `LdapCaCertMap` for CA certificates (see "LdapCaCertMap Mapper" on page 737)

Figure 21.2 Default mappers created during installation



It is important that you review and customize these mappers. For instructions on modifying mappers or creating new mappers, see "Step 3. Configure the Certificate Manager to Publish Certificates" on page 810.

LdapCaSimpleMap Module

The Java class that implements the CA certificate mapper is as follows:

`com.netscape.certsrv.ldap.LdapCaSimpleMap`

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.mapper.impl.LdapCaSimpleMap.class=com.netscape.certsrv.ldap.LdapCaSimpleMap`
- In the CMS window, the module is identified as follows: `LdapCaSimpleMap`

Figure 21.3 shows how the configurable parameters for the `LdapCaSimpleMap` module are displayed in the CMS window.

Figure 21.3 Parameters defined in the LdapCaSimpleMap module

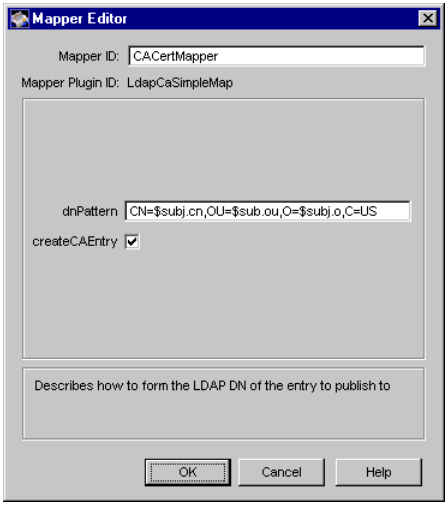


Table 21.2 describes these parameters.

Table 21.2 Description of parameters defined in the LdapCaSimpleMap module

Parameter	Description
`dnPattern`	Specifies the DN pattern the Certificate Manager should use to construct the DN in order to search for the CA's entry in the publishing directory. The value of `dnPattern` can be a list of AVA s separated by commas. An AVA can be a variable, such as `CN=$subj.cn`, that the Certificate Manager can derive from the certificate subject name, or a constant, such as `O=Siroe Corporation`. Permissible values: A valid pattern that will enable the Certificate Manager to construct the DN for the CA's entry. Example: `CN=$subj.cn,OU=$subj.ou,O=$subj.o,C=US`
`createCAEntry`	Specifies whether the Certificate Manager should create an entry for the CA in the publishing directory. Check the box if you want the server to create a CA's entry (default). Uncheck the box if you don't want the server to create an entry. If you check the box, the Certificate Manager first attempts to create an entry for the CA in the directory. If the Certificate Manager succeeds in creating the entry, it then attempts to publish the CA's certificate to the entry. Note that the CA's entry DN in the directory will match the pattern you specify in the `dnPattern` field. For example, if the issuer DN (specified in the CA's signing certificate) is `CN=testCA, OU=Research Dept, O=Siroe Corporation, C=US`, and the `dnPattern` is set to `CN=$subj.cn,OU=$subj.ou,O=$subj.o,C=US`, the Certificate Manager creates an entry with `CN=testCA, OU=Research Dept, O=Siroe Corporation, C=US` as its DN.

LdapCaCertMap Mapper

The mapper named `LdapCaCertMap` is an instance of the `LdapCaSimpleMap` module. The Certificate Manager automatically creates this mapper during installation.

You can use this mapper for creating an entry for the CA in the directory and for mapping the CA certificate to the CA's entry in the directory.

By default, the mapper is configured to create an entry for the CA in the directory and the default DN pattern for locating the CA's entry is as follows:

```
UID=$subj.cn,OU=people,O=$subj.o
```

LdapCrlMap Mapper

The mapper named `LdapCrlMap` is an instance of the `LdapSimpleMap` module. The Certificate Manager automatically creates this mapper during installation.

You can use this mapper for creating an entry for the CA in the directory and for mapping the CRL to the CA's entry in the directory.

By default, the mapper is configured to create an entry for the CA in the directory and the default DN pattern for locating the CA's entry is as follows:

```
UID=$subj.cn,OU=people,O=$subj.o
```

DN Components Mapper

The `LdapDNCompsMap` plug-in module implements the DN components mapper. This mapper enables you to configure a Certificate Manager to map a certificate to an LDAP directory entry by constructing the entry's distinguished name from components (such as `CN`, `OU`, `O`, and `C`) specified in the certificate subject name, and then using it as the search DN to locate the entry in the directory. You can use this mapper to locate the following:

- The CA's entry in the directory for publishing the CA certificate and the CRL.
- End-entity entries in the directory for publishing end-entity certificates.

The mapper requires you to specify values for three parameters, `filterComps`, `dnComps`, and `baseDN`, which are explained in Table 21.3. In general, the mapper takes DN components to build the search DN. The mapper also takes an optional root search DN. The server uses the DN components to form an LDAP entry to begin a subtree search and the filter components to form a search filter for the subtree. If none of the DN components are configured, the server uses the base DN for the subtree. If the base DN is null and none of the DN components match, an error is returned. If none of the DN components and filter components match, an error is returned. If the filter components are null, a base search is performed.

Note that both `DNComps` and `filterComps` parameters accept valid DN components or attributes separated by commas. The parameters don't accept multiple entries of an attribute; for example, you can set `filterComps` to `CN,OU`, but not to `CN,OU2,OU1`. If there's a need for you to support such a filter,

for example, if your directory entries contain multiple `OU`s and you want to use multiple `OU`s in your `filterComps` for filtering entries, you can modify the source code for the `LdapDNCompsMap` module. The java class for the module is in this directory:

```
<server_root>/cms_sdk/samples/publishing/mappers
```

The discussion below explains how mapping by DN components works. It is recommended that you read this before configuring a Certificate Manager to use this mapper.

Subject names in certificates are in distinguished-name format. A *distinguished name* (DN) uniquely identifies an entry in an LDAP directory. The DN consists of components that help identify the entry; for details, see Appendix A, "Distinguished Names."

The following components are commonly used in DNs:

- `UID`, which represents the user ID of a user in the directory
- `CN`, which represents the common name of a user in the directory
- `OU`, which represents an organizational unit in the directory
- `O`, which represents an organization in the directory
- `L`, which represents a locality in the directory
- `ST`, which represents a state in the directory
- `C`, which represents a country in the directory

For example, the following DN represents the user named Jane Doe who works for the Sales department at Siroe Corporation, which is located in Mountain View in the state of California, United States:

```
CN=Jane Doe, E=jdoe@siroe.com, OU=Sales, O=Siroe Corporation,
L=Mountain View, ST=California, C=US
```

The Certificate Manager uses the components in subject names to construct a DN that it can use as the *base* for searching specific directory entries in order to publish the corresponding certificate information.

For example, suppose the subject name in the certificate is in this form:

```
CN=Jane Doe, OU=Sales, O=Siroe Corporation, L=Mountain View,
ST=California, C=US
```

The Certificate Manager can use some or all of these components (`CN`, `OU`, `O`, `L`, `ST`, and `C`) to build a DN for searching the directory. When creating a mapper rule, you can specify the components the server should use to build a DN (that is, components to match attributes in the directory). You do this by configuring the `dnComps` parameter; for details, see Table 21.3 on page 742.

For example, assume you entered components `CN`, `E`, `OU`, `O`, and `C` as values for the `dnComps` parameter. For locating Jane Doe's entry in the directory, the Certificate Manager constructs the following DN by reading the DN attribute values from the certificate, and uses the DN as the base for searching the directory:

```
CN=Jane Doe, OU=Sales, O=Siroe Corporation, C=US
```

Note the following:

- A subject name does not need to have all of the components that you specify for the `dnComps` parameter. The server ignores any components that are not part of the subject name (such as `L`, `ST`, and `E` in this example).
- Unspecified components are not used to build the DN. In the example, if you did not include the `OU` component, the server would use this DN as the base for searching the directory:

  ```
  CN=Jane Doe, O=Siroe Corporation, C=US
  ```

In general, for the `dnComps` parameter, you should enter those DN components that the Certificate Manager can use to form the LDAP DN exactly. In certain situations, however, the subject name in a certificate may match more than one entry in the directory. Then, the Certificate Manager might not get a single, distinct matching entry from the DN. For example, the subject name

```
CN=Jane Doe, OU=Sales, O=Siroe Corporation, C=US
```

might match two users with the name Jane Doe in the directory. If that occurred, the Certificate Manager would need additional criteria to determine which entry corresponds to the subject of the certificate.

To specify the components the Certificate Manager must use to distinguish between different entries in the directory, use the `filterComps` parameter; for details, see Table 21.3 on page 742.

For example, if you entered `CN`, `OU`, `O`, and `C` as values for the `dnComps` parameter, enter `L` for the `filterComps` parameter only if the `L` attribute can be used to distinguish between entries with identical `CN`, `OU`, `O`, and `C` values.

Consider another example that shows how two directory entries with similar DNs can be differentiated by the value of the `UID` attribute: Assume that the two Jane Doe entries are distinguished by the value of the `UID` attribute. One entry's `UID` value is `janedoe1` and the other entry's `UID` value is `janedoe2`. Because the `UID` attribute corresponds to the `UID` component in a DN, you can set up the subject names of certificates to include the `UID` component.

Note Generally, the `E`, `L`, and `ST` components are not included in the standard set of certificate request forms provided for end entities. You can add these components to the forms, or you can have the issuing agents insert these components when editing the subject name in the certificate issuance forms.

LdapDNCompsMap Module

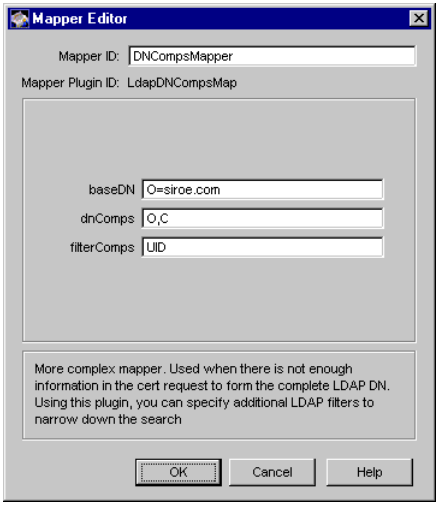
The Java class that implements the DN components mapper is identified as follows:

```
com.netscape.certsrv.ldap.LdapCertCompsMap
```

- In the configuration file, the plug-in module is identified as follows: `ca.publish.mapper.impl.LdapDNCompsMap.class=com.netscape.certsrv.ldap.LdapCertCompsMap`
- In the CMS window, the module is identified as follows: `LdapDNCompsMap`

Figure 21.4 shows how the configurable parameters for the `LdapDNCompsMap` module are displayed in the CMS window.

Figure 21.4 Parameters defined in the LdapDNCompsMap module



With this configuration, a Certificate Manager maps its certificates with the ones in the LDAP directory by using the `dnComps` values to form a DN and the `filterComps` values to form a search filter for the subtree.

- If the formed DN is null, the server uses the `baseDN` value for the subtree. If both the formed DN and base DN are null, the server logs an error.
- If the filter is null, the server uses the `baseDN` value for the search. If both the filter and base DN are null, the server logs an error.

Table 21.3 describes these parameters.

Table 21.3 Description of parameters defined in the LdapDNCompsMap module

Parameter	Description
`baseDN`	Specifies the DN to start searching for an entry in the publishing directory. If you leave the `dnComps` field blank, the server uses the base DN value to start its search in the directory. Permissible values: Alphanumeric string up to 255 characters; see “Base Distinguished Name” on page 1156. Example: `O=siroe.com`

Table 21.3 Description of parameters defined in the LdapDNCompsMap module (Continued)

Parameter	Description
`dnComps`	Specifies where in the publishing directory the Certificate Manager should start searching for an LDAP entry that matches the CA's or the end entity's information (that is, the owner of the certificate). The server uses the `dnComps` values to form an LDAP entry to begin a subtree search. The server gathers values for these attributes from the certificate subject name and uses the values to form an LDAP DN, which then determines where in the LDAP directory the server starts its search. For example, if you set `dnComps` to use the `O` and `C` attributes of the DN, the server starts the search from the `O=<org>`, `C=<country>` entry in the directory, where `<org>` and `<country>` are replaced with values from the DN in the certificate. If you leave the `dnComps` field empty, the server checks the `baseDN` field and searches the directory tree specified by that DN for entries matching the filter specified by `filterComps` parameter values. Permissible values: Valid DN components or attributes separated by commas. Example: `O,C`
`filterComps`	Specifies components the Certificate Manager should use to filter entries from the search result. The server uses the `filterComps` values to form an LDAP search filter for the subtree. The server constructs the filter by gathering values for these attributes from the certificate subject name; it uses the filter to search for and match entries in the LDAP directory. If the server finds one or more entries in the LDAP directory that match the information gathered from the certificate, the search is successful and the server optionally performs a verification. For example, if `filterComps` is set to use the email and user ID attributes (`filterComps=e`, `uid`), the server searches the directory for an entry whose values for email and user ID match the information gathered from the certificate. Email addresses and user IDs are good filters because they are usually unique entries in the directory. Keep in mind that email is not always included in the certificate subject name. The filter needs to be specific enough to match one and only one entry in the LDAP database. Permissible values: Valid directory attributes (in the certificate DN) separated by commas. The attribute names for the filters need to be attribute names from the certificate, not from ones in the LDAP directory. For example, most certificates have an `E` attribute for the user's email address; LDAP calls that attribute `mail`. Example: `UID`

Subject Name Mapper

The `LdapDNExactMap` plug-in module implements the subject name mapper. This mapper enables you to configure a Certificate Manager to map a certificate to an LDAP directory entry by searching for the LDAP entry DN that matches the certificate subject name. Note that to be able to use this mapper, each certificate subject name must exactly match a DN in a directory entry. For example, assume the certificate subject name is this:

```
UID=jdoe, O=Siroe Corporation, C=US
```

When searching the directory for the entry, the Certificate Manager only searches for an entry whose DN is this:

```
UID=jdoe, O=Siroe Corporation, C=US
```

If no matching entries are found, the server returns an error and does not publish the certificate.

This mapper does not require you to specify any values for any parameters because it obtains all values from the certificate (see Figure 21.5).

LdapDNExactMap Module

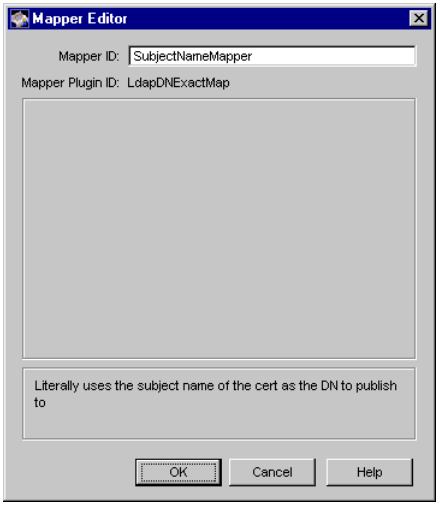
The Java class that implements the subject name mapper is identified as follows:

```
com.netscape.certsrv.ldap.LdapCertExactMap
```

- In the configuration file, the plug-in module is identified as follows: `ca.publish.mapper.impl.LdapDNExactMap.class=com.netscape.certsrv.ldap.LdapCertExactMap`
- In the CMS window, the module is identified as follows: `LdapDNExactMap`

Figure 21.5 shows how the `LdapDNExactMap` module looks when viewed in the CMS window.

Figure 21.5 The LdapDNExactMap module



Simple Mapper

The `LdapSimpleMap` plug-in module implements the simple mapper. This mapper enables you to configure a Certificate Manager to map a certificate to an LDAP directory entry by formulating the entry's DN from components specified in the certificate's subject name and attribute variable assertion (AVA) constants. For more information on AVAs, see the directory documentation.

The simple mapper requires you to specify just one parameter, which is named `dnPattern`. The value of `dnPattern` can be a list of AVA s separated by commas. An AVA can be a variable, such as `UID=$subj.UID`, or a constant, such as `O=Siroe Corporation`. By default, the Certificate Manager uses mapper rules that are based on the simple mapper.

During installation, the Certificate Manager automatically creates an instance (called a mapper) of the simple mapper module. The mapper is named `LdapUserCertMap` (see Figure 21.2 on page 735). You can use the default mapper to map various types of end-entity certificates the server will issue to their corresponding directory entries. For details, see "LdapUserCertMap Mapper" on page 747.

It is important that you review and customize this mapper. For instructions on modifying mappers or creating new mappers, see “Step 3. Configure the Certificate Manager to Publish Certificates” on page 810.

LdapSimpleMap Module

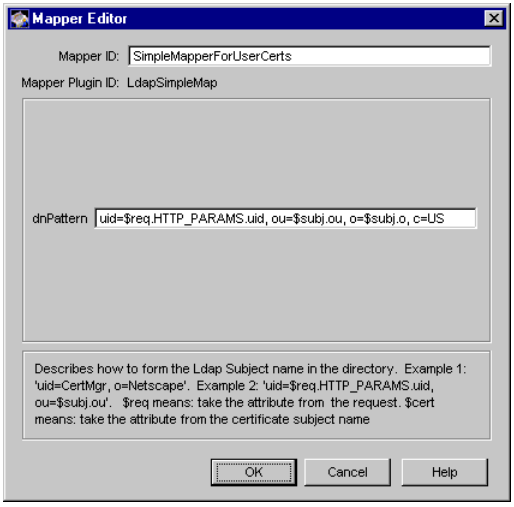
The Java class that implements the simple mapper is as follows:

`com.netscape.certsrv.ldap.LdapSimpleMap`

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.mapper.impl.LdapSimpleMap.class=com.netscape.certsrv.ldap.LdapSimpleMap`
- In the CMS window, the module is identified as follows: `LdapSimpleMap`

Figure 21.6 shows how configurable parameters for the `LdapSimpleMap` module are displayed in the CMS window.

Figure 21.6 Parameters defined in the LdapSimpleMap module



LdapUserCertMap Mapper

The rule named `LdapUserCertMap` is an instance of the `LdapSimpleMap` module. The Certificate Manager automatically creates this mapper during installation.

You can use this mapper for mapping end-user certificates to users' directory entries. The default DN pattern for locating end-user entries is as follows:

```
UID=$subj.UID, OU=people, O=$subj.o
```

The default pattern indicates that the Certificate Manager should use the `UID` and `O` values from the certificate subject name and a constant `OU=people` to construct the DN pattern in order to search for an entry.

For example, if the certificate subject name is

```
CN=Jane Doe, UID=jdoe, OU=people, O=Siroe Corporation, C=US
```

the Certificate Manager will construct the following DN to search the directory for the entry:

```
UID=jdoe, OU=people, O=Siroe Corporation
```

Subject Attribute Mapper

The `LdapSubjAttrMap` plug-in module implements the subject attribute mapper. This mapper enables you to configure a Certificate Manager to map a certificate to an LDAP directory entry by using the LDAP attribute named `certSubjectDN`. Note that for you to be able to use this mapper, your directory entries must include the `certSubjectDN` attribute.

This mapper requires you to specify the exact pattern of the subject DN because the Certificate Manager searches the directory for the `certSubjectDN` attribute whose value exactly matches the entire subject DN specified in the mapper configuration. For example, assume the certificate subject name is this:

```
UID=jdoe, O=Siroe Corporation, C=US
```

When searching the directory for the entry, the Certificate Manager first searches for entries that have these attributes in common

```
certSubjectDN=UID=jdoe, O=Siroe Corporation, C=US
```

and then narrows down the search to an entry that has only this:

```
certSubjectDN=UID=jdoe, O=Siroe Corporation, C=US
```

If no matching entries are found, the server returns an error and writes it to the log; see "Monitoring Logs" on page 1064.

LdapSubjAttrMap Module

The Java class that implements the subject attribute mapper is identified as follows:

`com.netscape.certsrv.ldap.LdapCertSubjMap`

- In the configuration file, the plug-in module is identified as follows: `ca.publish.mapper.impl.LdapSubjAttrMap.class=com.netscape.certsrv.ldap.LdapCertSubjMap`
- In the CMS window, the module is identified as follows: `LdapSubjAttrMap`

Figure 21.7 shows how configurable parameters for the `LdapSubjAttrMap` module are displayed in the CMS window.

Figure 21.7 Parameters defined in the LdapSubjAttrMap module

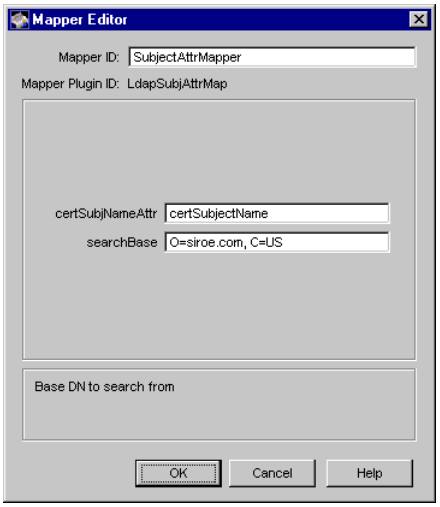


Table 21.4 describes these parameters.

Table 21.4 Description of parameters defined in the LdapSubjAttrMap module

Parameter	Description
`certSubjNameAttr`	Specifies the name of the LDAP attribute that contains a certificate subject name as its value. Permissible values: Must be `certSubjectName`. Example: `certSubjectName`
`searchBase`	Specifies the base DN for starting the attribute search. Permissible values: A valid DN of an LDAP entry. Example: `O=siroe.com, C=`US

Publisher Modules

Publisher modules help you configure the Certificate Manager to publish a CA certificate, end-entity certificates, or CRLs to the following:

- A mapped entry in the directory (entries are mapped by one of the mapper modules explained in "Mapper Modules" on page 732)
- A particular file
- An online validation authority

Overview of Publisher Modules

By default, the Certificate Manager provides publisher modules for publishing the CA certificate, end-entity certificates, and CRLs. Plug-in modules are implemented as Java classes and are registered in the CMS publishing framework. The Publisher Plugin Registration tab of the CMS window (see Figure 21.8) lists all the modules and the corresponding classes that are currently registered with a Certificate Manager.

Figure 21.8 Default publisher modules registered with a Certificate Manager

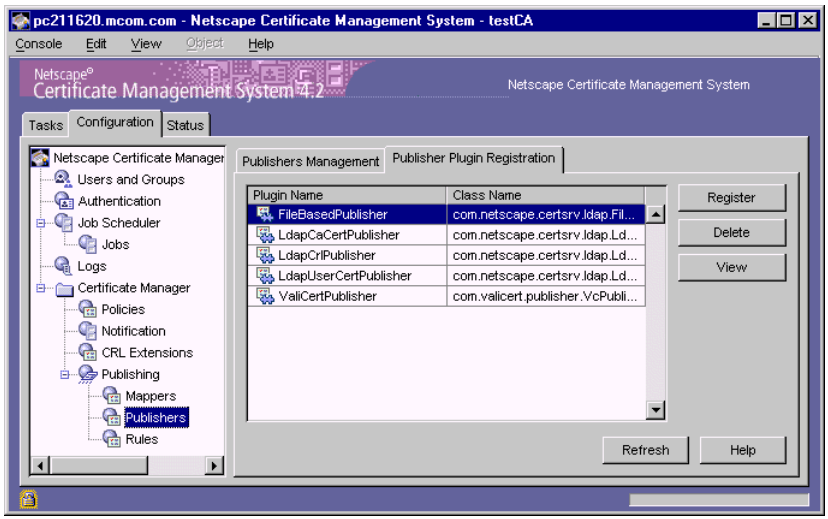


Table 21.5 describes the publisher modules provided for the Certificate Manager. You can use these modules to configure a Certificate Manager to employ specific publishing rules.

Table 21.5 Default publisher plug-in modules for publishing certificates and CRLs

Plug-in module name	Function
`FileBasedPublisher`	Publishes certificates and CRLs to a flat file (for exporting into other repositories). For details, see "Flat File Publisher" on page 752.
`LdapCaCertPublisher`	Publishes or unpublishes a certificate to the `caCertificate;binary` attribute of the mapped directory entry as a DER encoded binary blob. Also converts the object class to a `certificationAuthority` if it's not one already; similarly, removes the `certificationAuthority` object class on unpublish if the CA has no other certificates. For details, see "CA Certificate Publisher" on page 753.
`LdapCrlPublisher`	Publishes (replaces) a CRL to the `certificateRevocationList;binary` attribute of the mapped directory entry as a DER encoded binary blob. The entry should be a `certificationAuthority` object class. For details, see "CRL Publisher" on page 757.

Table 21.5 Default publisher plug-in modules for publishing certificates and CRLs (Continued)

Plug-in module name	Function
`LdapUserCertPublisher`	Publishes or unpublishes a certificate to the `userCertificate;binary` attribute of the mapped directory entry as a DER encoded binary blob. For details, see "End-Entity Certificate Publisher" on page 755.
`ValiCertPublisher`	Publishes the CRL to ValiCert Certificate VA™. For details, see "ValiCert Publisher" on page 759.

If you determine that the default publisher modules do not meet your requirements, you can develop a custom publisher class by implementing the following Java interface:

```
com.netscape.certsrv.ldappublish.ILdapPublisher
```

For more information on this interface, check the CMS software development kit (SDK) installed at this location: `<server_root>/cms_sdk/sdkdocs`

Be sure to take a look at the samples available at this location: `<server_root>/cms_sdk/samples/publishing/publishers`

When developing a custom publisher module, you may want to intercept LDAP error 52 and reword it so that the correct error message gets logged. To give you an example, if the publishing directory has been stopped, the server logs the following message in its error and system logs:

```
Error publishing CRL MasterCRL: Cannot find a match in the LDAP
server for certificate. netscape.ldap.LDAPException: unable to
establish connection (52); DSA is unavailable.
```

Notice that the error message incorrectly says DSA is unavailable instead of Directory Server is unavailable.

For instructions on how to configure a Certificate Manager to use a publisher module, see "Step 3. Configure the Certificate Manager to Publish Certificates" on page 810.

Flat File Publisher

The `FileBasedPublisher` plug-in module implements the flat file publisher. This module enables you to configure a Certificate Manager to publish certificates and CRLs to files, which then can be used for importing the certificates and CRLs into any other repository.

By default, the Certificate Manager does not create an instance of the `FileBasedPublisher` module. The instructions covered in "Publishing Certificates and CRLs to Flat Files" on page 840 explain how to create an instance of this module and how to configure a Certificate Manager to publish certificates and CRLs to files.

FileBasedPublisher Module

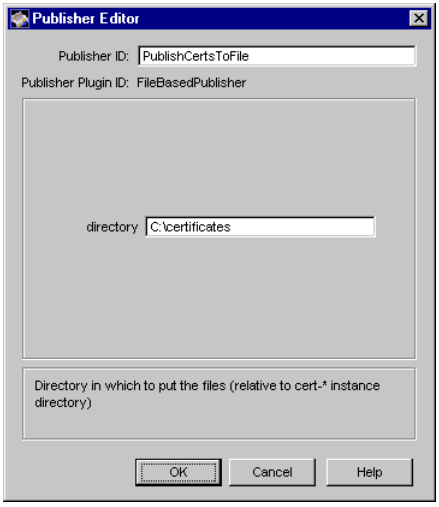
The Java class that implements the flat file publisher is as follows:

```
com.netscape.certsrv.ldap.FileBasedPublisher
```

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.publisher.impl.FileBasedPublisher.class=com.netscape.certsrv.ldap.FileBasedPublisher`
- In the CMS window, the module is identified as follows: `FileBasedPublisher`

Figure 21.9 shows how the configurable parameters for the `FileBasedPublisher` module are displayed in the CMS window.

Figure 21.9 Configuration parameters defined in the FileBasedPublisher module



The configuration shown in Figure 21.9 creates a publisher named `PublishCertsToFile`, which can publish certificate and CRL files to a directory at `C:\certificates`.

CA Certificate Publisher

The `LdapCaCertPublisher` plug-in module implements the CA certificate publisher. This module enables you to configure a Certificate Manager to publish or unpublish a certificate to the `caCertificate;binary` attribute of the mapped directory entry; the mapper must locate the correct entry so the publisher can publish the certificate to the specified attribute. The certificate is published as a DER encoded binary blob.

The module also converts the object class of the CA's entry to a `certificationAuthority` if it's not one already. Similarly, it also removes the `certificationAuthority` object class on unpublish if the CA has no other certificates.

You can use this module for publishing the CA certificate to the LDAP directory only.

During installation, the Certificate Manager automatically creates an instance (called a publisher) of the `LdapCaCertPublisher` module for publishing the CA certificate to the directory. See "LdapCaCertPublisher Publisher" on page 755.

LdapCaCertPublisher Module

The Java class that implements the CA certificate publisher is as follows:

```
com.netscape.certsrv.ldap.LdapCaCertPublisher
```

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.publisher.impl.LdapCaCertPublisher.class=com.netscape.certsrv.ldap.LdapCaCertPublisher`
- In the CMS window, the module is identified as follows: `LdapCaCertPublisher`

Figure 21.10 shows how the configurable parameters for the `LdapCaCertPublisher` module are displayed in the CMS window.

Figure 21.10 Parameters defined in the LdapCaCertPublisher module

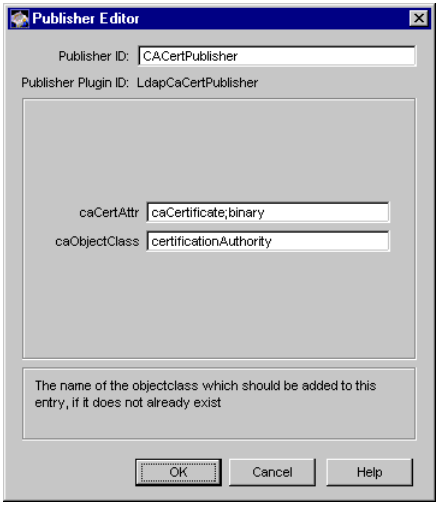


Table 21.6 describes these parameters.

Table 21.6 Description of parameters defined in the LdapCaCertPublisher module

Parameter	Description
`caCertAttr`	Specifies the LDAP directory attribute to publish the CA certificate. Permissible values: Must be `caCertificate;binary`. Example: `caCertificate;binary`
`caObjectClass`	Specifies the object class for the CA's entry in the directory. Permissible values: Must be `certificationAuthority`. Example: `certificationAuthority`

LdapCaCertPublisher Publisher

The publisher named `LdapCaCertPublisher` is an instance of the `LdapCaCertPublisher` module. The Certificate Manager automatically creates this publisher during installation.

You can use this publisher for publishing the CA certificate to `caCertificate;binary` attribute of the mapped CA's entry in the directory.

End-Entity Certificate Publisher

The `LdapUserCertPublisher` plug-in module implements the end-entity certificate publisher. This module enables you to configure a Certificate Manager to publish or unpublish a certificate to the `userCertificate;binary` attribute of the mapped directory entry; the mapper must locate the correct entry so the publisher can publish the certificate to the specified attribute. The certificate is published as a DER encoded binary blob.

You can use this module to publish any end-entity certificate to an LDAP directory. Types of end-entity certificates include SSL client, S/MIME, SSL server, object signing, router, and OCSP responder.

During installation, the Certificate Manager automatically creates an instance (called a publisher) of the `LdapUserCertPublisher` module for publishing end-entity certificates to the directory. See "LdapUserCertPublisher Publisher" on page 757.

LdapUserCertPublisher Module

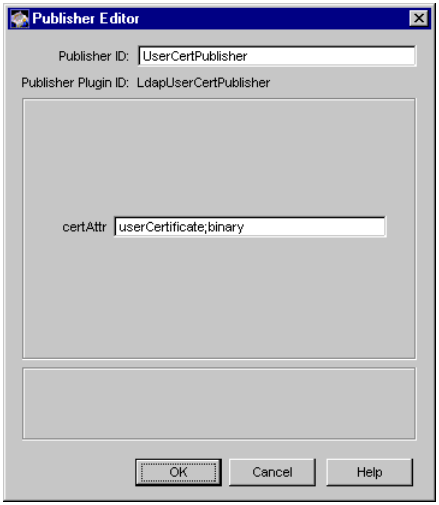
The Java class that implements the end-entity certificate publisher is as follows:

```
com.netscape.certsrv.ldap.LdapUserCertPublisher
```

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.publisher.impl.LdapUserCertPublisher.class=com.netscape.certsrv.ldap.LdapUserCertPublisher`
- In the CMS window, the module is identified as follows: `LdapUserCertPublisher`

Figure 21.11 shows how the configurable parameters for the `LdapUserCertPublisher` module are displayed in the CMS window.

Figure 21.11 Parameters defined in the LdapUserCertPublisher module



The configuration shown in Figure 21.11 creates a publisher rule named `LdapUserCertPublisher`, which publishes user certificates to the `userCertificate;binary` attribute of the mapped user entries.

Table 21.7 describes the parameters.

Table 21.7 Description of parameters defined in the LdapUserCertPublisher module

Parameter	Description
`certAttr`	Specifies the directory attribute of the mapped entry to which the Certificate Manager should publish the certificate. Permissible values: Must be `userCertificate;binary`. Example: `userCertificate;binary`

LdapUserCertPublisher Publisher

The publisher named `LdapUserCertPublisher` is an instance of the `LdapUserCertPublisher` module. The Certificate Manager automatically creates this publisher during installation.

You can use this publisher to publish an end-entity certificate to the `userCertificate;binary` attribute of the mapped end-entity's entry in the directory.

CRL Publisher

The `LdapCrlPublisher` plug-in module implements the CRL publisher. This module enables you to configure a Certificate Manager to publish or unpublish the CRL to the `certificateRevocationList;binary` attribute of the mapped directory entry; the configured mapper must locate the CA's entry so that the publisher can publish the CRL to the `certificateRevocationList;binary` attribute. The CRL is published as a DER-encoded binary blob.

The CRL publisher requires you to specify just one parameter named `crlAttr`. The value of this parameter must be `certificateRevocationList;binary`.

During installation, the Certificate Manager automatically creates an instance (called a publisher) of the `LdapCrlPublisher` module for publishing CRLs to the directory. See "LdapCrlPublisher Publisher" on page 759.

LdapCrlPublisher Module

The Java class that implements the CRL publisher is as follows:

`com.netscape.certsrv.ldap.LdapCrlPublisher`

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.publisher.impl.LdapCrlPublisher.class=com.netscape.certsrv.ldap.LdapCrlPublisher`
- In the CMS window, the module is identified as follows: `LdapCrlPublisher`

Figure 21.12 shows how the configurable parameters for the `LdapCrlPublisher` module are displayed in the CMS window.

Figure 21.12 Parameters defined in the LdapCrlPublisher module

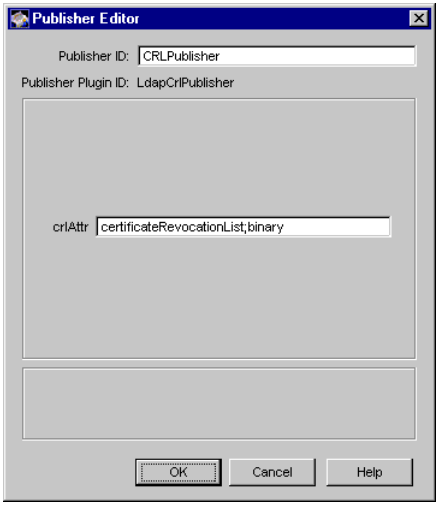


Table 21.8 describes these parameters.

Table 21.8 Description of parameters defined in the LdapCrlPublisher module

Parameter	Description
`crlAttr`	Specifies the directory attribute of the mapped entry to which the Certificate Manager should publish the certificate. Permissible values: Must be `certificateRevocationList;binary`. Example: `certificateRevocationList;binary`

LdapCrlPublisher Publisher

The publisher named `LdapCrlPublisher` is an instance of the `LdapCrlPublisher` module. The Certificate Manager automatically creates this publisher during installation.

You can use this publisher for publishing the CRL to `certificateRevocationList;binary` attribute of the CA's entry in the directory.

ValiCert Publisher

The `ValiCertPublisher` plug-in module implements the ValiCert publisher. This publisher enables you to configure a Certificate Manager to publish CRLs to ValiCert Certificate VA™ (Certificate VA), which can function as a local online validation authority for your PKI setup. For details, see "Local OCSP Support" on page 729.

By default, the Certificate Manager does not create any instance of the `ValiCertPublisher` module. The instructions covered in "Publishing CRLs to Online Validation Authority" on page 857 explain how to create an instance of this module and how to configure a Certificate Manager to publish CRLs to Certificate VA.

ValiCertPublisher Module

The Java class that implements the ValiCert publisher is as follows:

`com.valicert.publisher.VcPublisher`

- In the CMS configuration file, the plug-in module is identified as follows: `ca.publish.publisher.impl.ValiCertPublisher.class=com.valicert.publisher.VcPublisher`
- In the CMS window, the module is identified as follows: `ValiCertPublisher`

Figure 21.13 shows how the configurable parameters for the `ValiCertPublisher` module are displayed in the CMS window.

Figure 21.13 Parameters defined in the ValiCertPublisher module

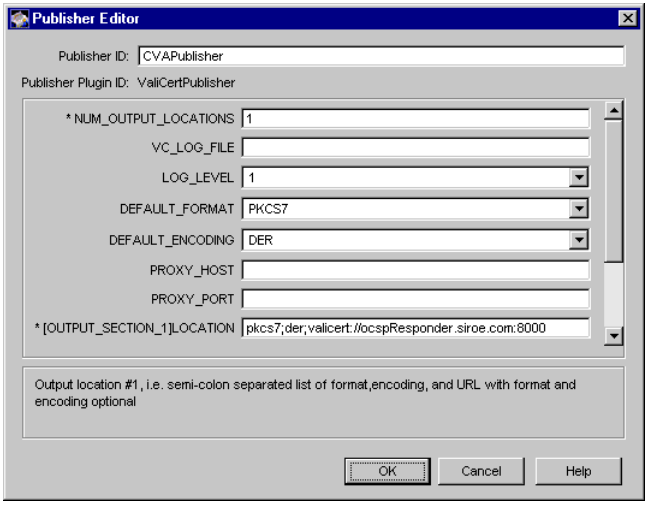


Table 21.9 describes these parameters.

Table 21.9 Description of parameters defined in the ValiCertPublisher module

Parameter	Description
NUM_OUTPUT_ LOCATIONS	Specifies the total number of output locations. You can change the total number of locations by changing the value assigned to this parameter. Permissible values: n (or leave the field blank). • n specifies the total number of locations; it must be an integer greater than zero. The default value is 5. • If you leave the value field blank, it defaults to 1. Example: 2
VC_LOG_FILE	Specifies the name of the file to which the server writes or logs messages to. If you leave the value field blank, the server writes to a file named ./ vpublish.log.
LOG_LEVEL	Specifies the log level, the level of information to be reported. Permitted values: 0, 1, or 2. • 0 specifies informational messages should be logged. • 1 specifies error and warning messages should be logged (default). • 1 specifies debugging messages should be logged. Example: 2
DEFAULT_FORMAT	Specifies the data format for the CRL; the value in this field is used if the CRL format is not specified by the OUTPUT_SECTION parameter. Permissible values: CRL or PKCS7. • CRL specifies that the CRL format should be the same as the one defined in the X.509 standard. • PKCS7 specifies that the CRL format should be PKCS #7 (default). Example: PKCS7

Table 21.9 Description of parameters defined in the ValiCertPublisher module (Continued)

Parameter	Description
DEFAULT_ENCODING	Specifies the encoding to be used for the CRL; the value in this field is used if the encoding is not specified by the OUTPUT_SECTION parameter. Permissible values: DER or BASE64. • DER specifies that the CRL should be published as a DER-encoded blob (default). • BASE64 specifies that the CRL should be published as a base-64 encoded blob. Example: DER
PROXY_HOST	Specifies the name of the proxy server. If you leave the field blank, proxy server will not be used.
PROXY_PORT	Specifies the port number of the proxy server. If you leave the field blank, proxy server will not be used.
[OUTPUT_SECTIONS_<n>]LOCATION	Specifies the location or the destination information to publish the CRL. Each location is distinguished by <n>, which is an integer derived from the value you assigned in the NUM_OUTPUT_LOCATIONS field. For example, if you set the NUM_OUTPUT_LOCATIONS parameter to 2, <n> would be 1 and 2. The syntax for specifying the destination information is as follows: [<format>;][<encoding>;]valicert://<host>[:<port>][/<location>] Attribute inside the square bracket, [], is optional. Permissible values: • For <format>: CRL or PKCS7 • For <encoding>: DER or BASE64 • For <host>: A fully qualified host name of the ValiCert Certificate VA. • For <port>: The service port of ValiCert Certificate VA; by default it is 80. Example: PKCS7;DER;valicert://validator.siroe.com:8000

CRL Extension Modules

Since its initial publication, the X.509 standard for CRL formats has been amended to include additional information within a CRL. Version 2, the latest version, allows you to add information as CRL extensions.

The extensions defined by ANSI X9 and ISO/IEC/ITU for X.509 v2 CRLs [X.509] [X9.55] enable you to associate additional attributes with CRLs. The *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* (see `http://www.ietf.org/rfc/rfc2459.txt`) recommends a set of extensions to be used in CRLs. These extensions are called *standard CRL extensions*.

The standard also suggests that you can define your own extensions and include them in CRLs you issue. These extensions are called *private*, *proprietary*, or *custom* CRL extensions and they carry information unique to your organization or business. Keep in mind that applications may not able to validate CRLs that contain private, critical extensions, thus preventing the use of these CRLs in a general context.

Note Some explanations in this chapter make reference to Abstract Syntax Notation One (ASN.1) and Distinguished Encoding Rules (DER). These are specified in the CCITT Recommendations X.208 and X.209. For a quick summary of ASN.1 and DER, see *A Layman's Guide to a Subset of ASN.1, BER, and DER*, which is available at RSA Laboratories' web site (`http://www.rsa.com`).

Structure of CRL Extensions

A CRL extension consists of the following:

- The object identifier (OID) for the extension; see "Object Identifier" on page 553.

  This identifier uniquely identifies the extension. It also determines the ASN.1 type of value in the value field and how the value is interpreted. That is, when an extension appears in a CRL, the OID appears as the extension ID field (`extnID`) and the corresponding ASN.1 encoded structure appears as the value of the octet string (`extnValue`); see the examples in "Sample CRL and CRL Entry Extensions" on page 764.

- A flag or boolean field called `critical`.

The `true` or `false` value assigned to this field indicates whether the extension is critical (true) or noncritical (false) to the CRL.

— If the extension is critical and the CRL is sent to an application that does not understand the extension (based on the extension's ID), the application must reject the CRL.

— If the extension is not critical and the CRL is sent to an application that does not understand the extension (based on the extension's ID), the application can ignore the extension and accept the CRL.

- An octet string containing the DER encoding of the value of the extension.

  Typically, the application receiving the CRL checks the extension ID to determine if it can recognize the ID. If it can, it uses the extension ID to determine the type of value used.

Sample CRL and CRL Entry Extensions

The following is an example of the section of a CRL containing X.509 v2 extensions. (Certificate Management System can display CRLs in human-readable format, as shown here.) As shown in the example, CRL extensions appear in sequence and only one instance of a particular extension may appear in a particular CRL; for example, a CRL may contain only one authority key identifier extension. However, CRL-entry extensions appear in appropriate entries in the CRL.

```
Certificate Revocation List:
   Data:
      Version:  v2
...

Extensions:
   Identifier: Authority Key Identifier
      Critical: no
      Key Identifier:
      2c:22:c6:ae:4e:4b:91:c7:fb:4c:cc:ae:84:e8:aa:5b:46:6a:a0:ad

Extensions:
   Identifier: Revocation Reason - 2.5.29.21
      Critical: no
      Reason: Key_Compromise
      Serial Number: 0x12
      Revocation Date: Tuesday, December 15, 1998 5:20:42 AM

Extensions:
   Identifier: Revocation Reason - 2.5.29.21
```

```
       Critical: no
       Reason: CA_Compromise
       Serial Number: 0x11
       Revocation Date: Wednesday, December 16, 1998 4:51:54 AM

Extensions:
   Identifier: Revocation Reason - 2.5.29.21
       Critical: no
       Reason: Key_Compromise
       Serial Number: 0x10
       Revocation Date: Thursday, December 17, 1998 2:37:24 AM

Extensions:
   Identifier: Revocation Reason - 2.5.29.21
       Critical: no
       Reason: Affiliation_Changed
       Serial Number: 0xA
       Revocation Date: Wednesday, November 25, 1998 5:11:18 AM

...
```

Overview of CRL Extension Modules

To enable you issue or publish X.509 v2 CRLs (that is, CRLs with extensions), Certificate Management System provides a set of plug-in modules; each module enables you to configure the Certificate Manager to set a particular CRL or CRL-entry extension in CRLs it issues. Plug-in modules are implemented as Java classes and are registered in the CMS publishing framework. The CRL Extensions Management tab of the CMS window (Figure 21.14) lists all the modules that are registered with a Certificate Manager.

When deciding whether to add CRL extensions, keep in mind that not all applications support version 2 CRLs. Among the applications that do support extensions, not all applications will recognize every extension. For general guidelines on using these extensions in CRLs, see "Certificate Extensions" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.14 Default CRL extension modules registered with a Certificate Manager

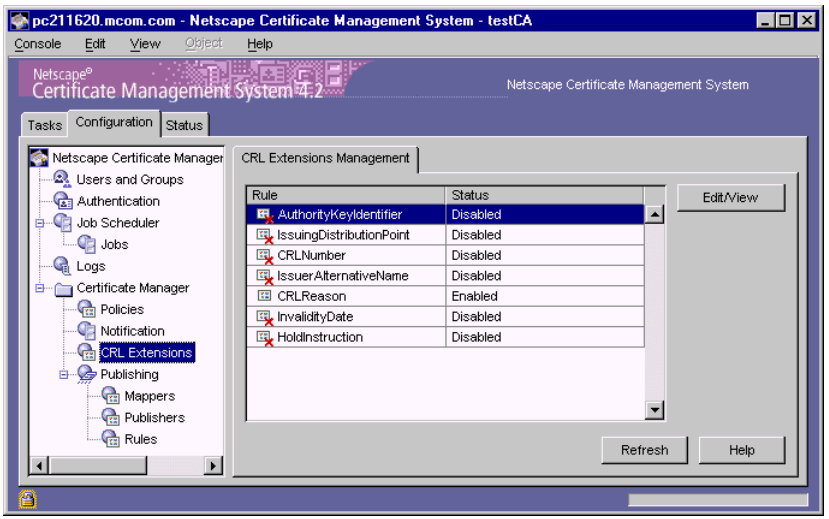


Table 21.10 lists CRL extension modules that are installed with a Certificate Manager.

Table 21.10 Default CRL extension modules

Plug-in module name	Function
`AuthorityKeyIdentifier`	Sets the *Authority Key Identifier* extension in CRLs. For details, see "AuthorityKeyIdentifier Rule" on page 767.
`CRLNumber`	Sets the *CRL Number* extension in CRLs. For details, see "CRLNumber Rule" on page 769.
`CRLReason`	Sets the *Reason Code* extension in CRL entries. For details, see "CRLReason Rule" on page 770.
`HoldInstruction`	Sets the *Hold Instruction Code* extension in CRL entries. For details, see "HoldInstruction Rule" on page 772.
`InvalidityDate`	Sets the *Invalidity Date* extension in CRL entries. For details, see "InvalidityDate Rule" on page 774.
`IssuerAlternativeName`	Sets the *Issuer Alternative Name* extension in CRLs. For details, see "IssuerAlternativeName Rule" on page 776.
`IssuingDistributionPoint`	Sets the *Issuing Distribution Point* extension in CRLs. For details, see "IssuingDistributionPoint Rule" on page 780.

For instructions on how to configure a Certificate Manager to set CRL extensions, see "Step B. Set the CRL Extensions" on page 826.

AuthorityKeyIdentifier Rule

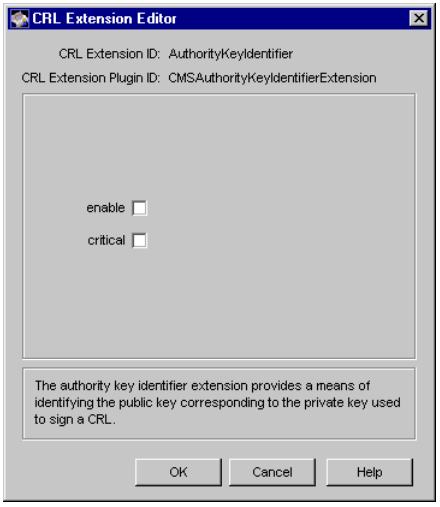
The `AuthorityKeyIdentifier` rule enables you to configure a Certificate Manager to set the *Authority Key Identifier Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. The extension is used to identify the public key that corresponds to the private key used by a CA to sign CRLs.

The PKIX standard recommends that the CA must include this extension in all CRLs it issues. Therefore, you should consider adding this extension to all CRLs issued by the Certificate Manager. The reason for this is that in certain situations, a CA's public key may change (for example, when the key gets updated) or the CA may have multiple signing keys (either because of multiple concurrent key pairs or because of key changeover). In these cases, the CA ends up with more than one key pair. When verifying a signature on a certificate, other applications need to know which key was used in the signature. The extension, if present in a certificate, enables applications (those that can use the extension) to identify the correct key to use in situations when multiple keys exist; the extension specifies the public key to be used to verify the signature on the CRL.

For general guidelines on setting the authority key identifier extension in CRLs, see "authorityKeyIdentifier" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.15 shows how configurable parameters for the `AuthorityKeyIdentifier` rule are displayed in the CMS window.

Figure 21.15 Parameters defined in the AuthorityKeyIdentifier rule



The configuration shown in Figure 21.15 specifies that the server should not set the authority key identifier extension in CRLs.

Table 21.11 describes these parameters.

Table 21.11 Description of parameters defined in the AuthorityKeyIdentifierExt rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the authority key identifier extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

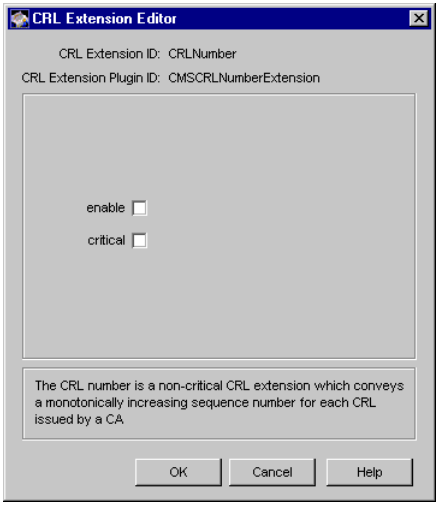
CRLNumber Rule

The `CRLNumber` rule enables you to configure a Certificate Manager to set the *CRL Number Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. This extension specifies a monotonically increasing sequence number for each CRL issued by a CA, allowing CRL users to easily determine when a particular CRL supersedes another CRL.

For general guidelines on setting the CRL number extension in CRLs, see "CRLNumber" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.16 shows how the configurable parameters for the `CRLNumber` rule are displayed in the CMS window.

Figure 21.16 Parameters defined in the CRLNumber rule



The configuration shown in Figure 21.16 specifies that the server should not set the CRL number extension in CRLs.

Table 21.12 describes these parameters.

Table 21.12 Description of parameters defined in the CRLNumber rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the CRL number extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

CRLReason Rule

The `CRLReason` rule enables you to configure a Certificate Manager to set the *CRL ReasonCode Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRL entries. The extension is used to identify the reason for the revocation of a certificate included in the CRL.

For general guidelines on setting the CRL reason code in CRL entries, see "reasonCode" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

The revocation reasons defined by the standard are listed in Table 21.13.

Table 21.13 Certificate revocation reasons

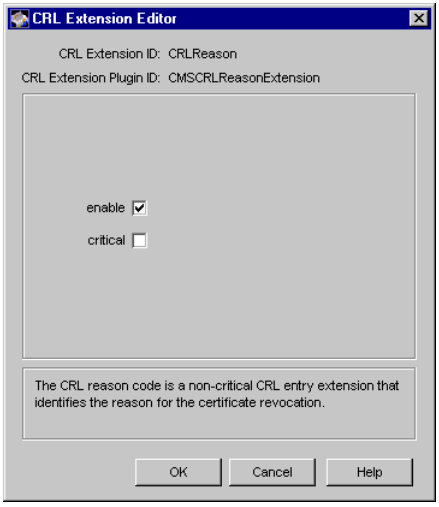
Code	Reason
0	`unspecified`
1	`keyCompromise`
2	`cACompromise`
3	`affiliationChanged`

Table 21.13 Certificate revocation reasons (Continued)

Code	Reason
4	`superseded`
5	`cessationOfOperation`
6	`certificateHold`
8	`removeFromCRL`

Figure 21.17 shows how the configurable parameters for the `CRLReason` rule are displayed in the CMS window.

Figure 21.17 Parameters defined in the CRLReason rule



The configuration shown in Figure 21.17 specifies that the server should set the CRL reason code extension in CRL entries.

Table 21.14 describes these parameters.

Table 21.14 Description of parameters defined in the CRLReason rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule (default). Uncheck the box to disable the rule. • If you enable the rule and set the remaining parameters correctly, the server sets the CRL number extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

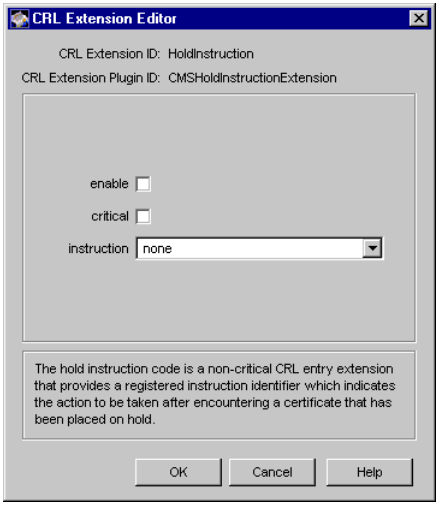
HoldInstruction Rule

The `HoldInstruction` rule enables you to configure a Certificate Manager to set the *CRL Hold Instruction Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. The extension is a non-critical CRL entry extension that is used to specify a registered instruction identifier—the identifier indicates what action the validating application should take when it encounters a certificate that has been placed on hold.

For general guidelines on setting the CRL hold instruction code in CRL entries, see "holdInstructionCode" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.18 shows how the configurable parameters for the `HoldInstruction` rule are displayed in the CMS window.

Figure 21.18 Parameters defined in the HoldInstruction rule



The configuration shown in Figure 21.18 specifies that the server should not set the hold instruction extension in CRL entries.

Table 21.15 describes these parameters.

Table 21.15 Description of parameters defined in the HoldInstruction rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the Hold Instruction extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

Table 21.15 Description of parameters defined in the HoldInstruction rule (Continued)

Parameter	Description
`instruction`	Specifies the action a validating application must take when it encounters a certificate that has been put on hold. Permissible values: `none`, `callissuer`, or `reject`. • `none` specifies that the validating application need not do anything; the PKIX standard says that this is semantically equivalent to the absence of a holdInstructionCode (default). • `callissuer` specifies that the validating application must call the CA that has issued the certificate or reject the certificate. • `reject` specifies that the validating application must reject the certificate on hold. Example: `none`

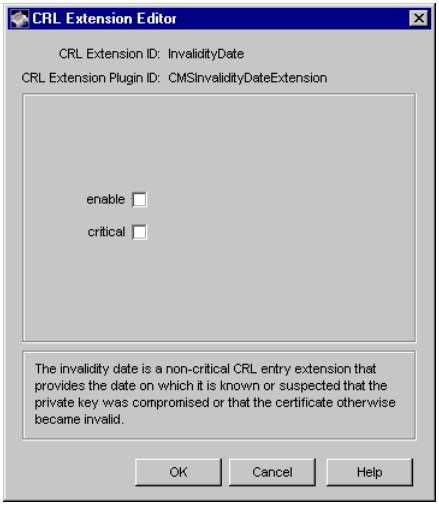
InvalidityDate Rule

The `InvalidityDate` rule enables you to configure a Certificate Manager to set the *Invalidity Date Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. The extension is a non-critical CRL entry extension that is used to specify the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid.

For general guidelines on setting the invalidity date extension in CRL entries, see "invalidityDate" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.19 shows how the configurable parameters for the `InvalidityDate` rule are displayed in the CMS window.

Figure 21.19 Parameters defined in the InvalidityDate rule



The configuration shown in Figure 21.19 specifies that the server should not set the invalidity date extension in CRL entries.

Table 21.16 describes these parameters.

Table 21.16 Description of parameters defined in the InvalidityDate rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the Invalidity Date extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

IssuerAlternativeName Rule

The `IssuerAlternativeName` rule enables you to configure a Certificate Manager to set the *Issuer Alternative Name Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. This extension enables binding of or associating alternative identities, such as Internet electronic mail address, a DNS name, an IP address, and a uniform resource indicator (URI), with the issuer of the CRL.

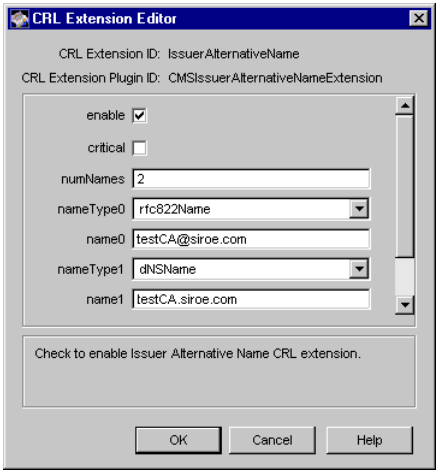
The `IssuerAlternativeName` rule enables you to associate the following identities with a CRL issuer, by including them in the extension:

- An rfc822Name
- A DNS name
- A directory name
- A uniform resource indicator (URI)
- An IP address
- An object identifier (OID)

For general guidelines on setting the issuer alternative name extension in CRLs, see "issuerAltName" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.20 shows how configurable parameters for the `IssuerAlternativeName` rule are displayed in the CMS window.

Figure 21.20 Parameters defined in the IssuerAlternativeName rule



The configuration shown in Figure 21.20 specifies that the server should not set the issuing point extension in CRLs.

Table 21.17 describes these parameters.

Table 21.17 Description of parameters defined in the IssuerAlternativeName rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the issuer alternative name extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical. Uncheck the box if you want the server to mark the extension noncritical (default).

Table 21.17 Description of parameters defined in the IssuerAlternativeName rule (Continued)

Parameter	Description
numNames	Specifies the total number of alternative names or identities permitted in the extension. Note that each name has a set of configuration parameters— nameType and name—and you must specify appropriate values for each of those parameters; otherwise, the policy rule will return an error. You can change the total number of identities by changing the value specified in this field; there's no restriction on the total number of identities you can include in the extension. Each set of configuration parameters is distinguished by <n>, which is an integer derived from the value you assign in this field. For example, if you set the numNames parameter to 2, <n> would be 0 and 1. Permissible values: 0 or n. • 0 specifies that no identities can be contained in the extension. • n specifies the total number of identities to be included in the extension; it must be an integer greater than zero. The default value is 3. Example: 1
nameType<n>	Specifies the general-name type. Permissible values: rfc822Name, directoryName, dNSName, ediPartyName, URL, iPAddress, OID, or otherName. • Select rfc822Name if the name is an Internet mail address. • Select directoryName if the name is an X.500 directory name. • Select dNSName if the name is a DNS name. • Select ediPartyName if the name is a EDI party name. • Select URL if the name is a uniform resource identifier (default). • Select iPAddress if the name is an IP address. • Select OID if the name is an object identifier. • Select otherName if the name is in any other name form. Example: URL

Table 21.17 Description of parameters defined in the IssuerAlternativeName rule (Continued)

Parameter	Description
name<n>	Specifies the general-name value. Permissible values: Depends on the name type specified in the nameType<n> field. • If the type is rfc822Name, the value must be a valid Internet mail address in the local-part@domain format; see the definition of an rfc822Name as defined in RFC 822 (http://www.ietf.org/rfc/rfc0822.txt). You may use upper and lower case letters in the mail address; no significance is attached to the case. Example: testCA@siroe.com • If the type is directoryName, the value must be a string form of X.500 name, similar to the subject name in a certificate, in the RFC 2253 syntax (see http://www.ietf.org/rfc/rfc2253.txt). Note that RFC 2253 replaces RFC 1779. Example: CN=CACentral,OU=Research Dept,O=Siroe Corp,C=US. • If the type is dNSName, the value must be a valid domain name in the preferred-name syntax as specified in RFC 1034 (http://www.ietf.org/rfc/rfc1034.txt). You may use upper and lower case letters in the domain name; no significance is attached to the case. Do not use the string " " for the DNS name. Also don't use the DNS representation for Internet mail addresses; such identities should be encoded as rfc822Name. Example: testCA.siroe.com • If the type is ediPartyName, the name must be an IA5String. Example: Siroe Corporation • If the type is URL, the value must be a non-relative universal resource identifier (URI) following the URL syntax and encoding rules specified in RFC 1738 (http://www.ietf.org/rfc/rfc1738.txt). That is, the name must include both a scheme (for example, http) and a fully qualified domain name or IP address of the host. Example: http://testCA.siroe.com

Table 21.17 Description of parameters defined in the IssuerAlternativeName rule (Continued)

Parameter	Description
	• If the type is `iPAddress`, the value must be a valid IP address specified in dot-separated numeric component notation. The syntax for specifying the IP address is as follows: For IP version 4 (IPv4), the address should be in the form specified in RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`). IPv4 address must be in the `n.n.n.n` format; for example, `128.21.39.40`. IPv4 address with netmask must be in the `n.n.n.n,m.m.m.m` format. For example, `128.21.39.40,255.255.255.00`. For IP version 6 (IPv6), the address should be in the form described in RFC 1884 (`http://www.ietf.org/rfc/rfc1884.txt`), with netmask separated by a comma. Examples of IPv6 addresses with no netmask are `0:0:0:0:0:0:13.1.68.3` and `FF01::43`. Examples of IPv6 addresses with netmask are `0:0:0:0:0:0:13.1.68.3,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:255.255.255.0` and `FF01::43,FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FF00:0000`.
	• If the type is `OID`, the value must be a unique, valid OID specified in the dot-separated numeric component notation. Although you can invent your own OIDs for the purposes of evaluating and testing this server, in a production environment, you should comply with the ISO rules for defining OIDs and for registering subtrees of IDs. See "Object Identifier" on page 553 for information on allocating private OIDs. Example: `1.2.3.4.55.6.5.99`
	• If the type is `otherName`, the name must be must be the absolute path to the file that contains the general name in its base-64 encoded format. Example: `C:\netscape\server4\extn\ian\othername.txt`

IssuingDistributionPoint Rule

The `IssuingDistributionPoint` rule enables you to configure a Certificate Manager to set the *Issuing Distribution Point Extension* defined in X.509 and PKIX standard RFC 2459 (see `http://www.ietf.org/rfc/rfc2459.txt`) in CRLs. The CRL issuing point extension enables you to specify a pointer to a

particular CRL and to include additional information about the CRL at that location—whether it covers revocation of end-entity certificates only, CA certificates only, or revoked certificates that have a limited set of reason codes.

By default, the pointer can be in either of these forms:

- The name of the X.500 directory that stores the CRL
- The URI to the location that contains the CRL

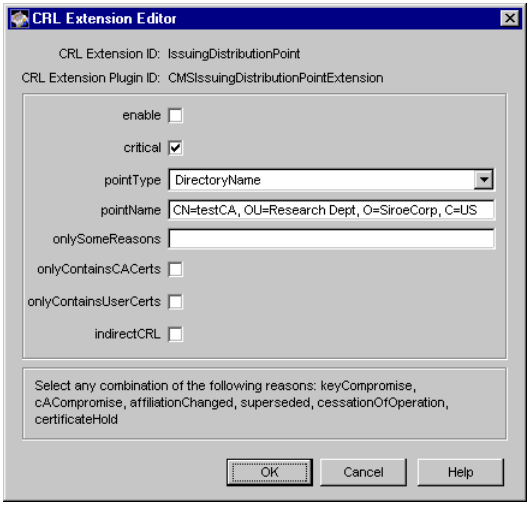
Optionally, each issuing point may contain a set of reason flags, indicating what revocation reasons are covered by the CRL at the specified location. Note that you can modify the rule to support any name form by making the appropriate changes to the sample code provided for this purpose. The sample code is located here:

```
<server_root>/cms_sdk/samples/CRLs/IssuingDistributionPoint
```

For general guidelines on setting the issuing distribution point extension in CRLs, see "issuingDistributionPoints" in Appendix B of *Netscape Certificate Management System Installation and Deployment Guide*.

Figure 21.21 shows how configurable parameters for the `IssuingDistributionPoint` rule are displayed in the CMS window.

Figure 21.21 Parameters defined in the IssuingDistributionPoint rule



The configuration shown in Figure 21.21 specifies that the server should not set the issuing point extension in CRLs.

Table 21.18 describes these parameters.

Table 21.18 Description of parameters defined in the IssuingDistributionPoint rule

Parameter	Description
`enable`	Specifies whether the rule is enabled or disabled. Check the box to enable the rule. Uncheck the box to disable the rule (default). • If you enable the rule and set the remaining parameters correctly, the server sets the issuing distribution point extension in CRLs. • If you disable the rule, the server does not add the extension to CRLs; it ignores the values in the remaining fields.
`critical`	Specifies whether the extension should be marked critical or noncritical in CRLs issued by the server. Check the box if you want the server to mark the extension critical (default). Uncheck the box if you want the server to mark the extension noncritical.
`pointType`	Specifies the type (for example, `URI`) of the issuing distribution point. Permissible values: By default, `DirectoryName` and `URI`. • `DirectoryName` specifies that the type is an X.500 Directory Name (that is, the CRL is stored in an X.500 directory). • URI specifies that the type is a uniform resource indicator (this provides a pointer to the location for the most current CRL issued by this CA). Example: `URI`

Table 21.18 Description of parameters defined in the IssuingDistributionPoint rule (Continued)

Parameter	Description
`pointName`	Specifies the name of the issuing distribution point. The name of the distribution point can be in any of the following formats: Permissible values: Depends on the value specified for the `pointType` parameter. • If the `pointType` attribute is set to `DirectoryName`, the name must be an X.500 Name (in RFC1779 syntax). • If the `pointType` attribute is set to `URI`, the name must be a URI; the URI must be an absolute pathname and must specify the host. Example: • If the name is a URI, it would look similar to this: `http://testCA.siroe.com/get/your/crls/here/` • If the name is an X.500 Directory Name, it would look similar to this: `CN=CRLCentral,OU=Research Dept,O=Siroe Corp,C=US` (Note that the CRL may be stored in the directory entry corresponding to the CRL issuing point, which may be different than the directory entry of the CA.)
`onlySomeReasons`	Specifies the reason codes associated with the distribution point. Permissible values: A combination of reason codes—`unspecified`, `keyCompromise`, `cACompromise`, `affiliationChanged`, `superseded`, `cessationOfOperation`, `certificateHold`, and `removeFromCRL`—separated by commas. Leave field blank if the distribution point contains revoked certificates with all reason codes or if you don't want to set this field (default). Example: `unspecified`, `keyCompromise`, `cessationOfOperation`
`onlyContainsCACerts`	Specifies whether the distribution point contains only revoked CA certificates. Check the box if the distribution point contains CA certificates only. Uncheck the box if the distribution point contains all types of revoked certificates (default).
`onlyContainsUserCerts`	Specifies whether the distribution point contains only revoked user certificates. Check the box if the distribution point contains user certificates only. Uncheck the box if the distribution point contains all types of certificates (default).

Table 21.18 Description of parameters defined in the IssuingDistributionPoint rule (Continued)

Parameter	Description
`indirectCRL`	Specifies whether the distribution point contains an indirect CRL. Check the box if the distribution point contains an indirect CRL. Uncheck the box if the distribution point doesn't contain an indirect CRL (default).

Chapter

22

Configuring a Certificate Manager for Publishing

Netscape Certificate Management System (CMS) provides a customizable publishing framework for the Certificate Manager. This chapter explains how to configure the Certificate Manager to publish to an LDAP directory, a flat file, and an online validation authority.

Before reading this chapter, you should have read the previous chapters in this part. In particular, you should be familiar with the mapper and publisher plug-in modules that are provided for the Certificate Manager. If you are not, see "Modules for Publishing Certificates and CRLs" on page 731.

The chapter has the following sections:

Publishing Certificates and CRLs to a Directory

If you are using an LDAP-compliant directory, such as Netscape Directory Server, to publish and manage your user and group data, you can configure the Certificate Manager to communicate with this directory. The Certificate Manager can then publish end-entity as well as CA certificates and the certificate revocation list (CRL) to the directory. This way, your publishing directory acts as a common distribution point for information about users and other entities on the network, including each entity's current security credentials.

Once the Certificate Manager is configured to publish to the directory, the following operations are performed automatically:

- When the Certificate Manager starts up, it publishes its CA certificate to the configured directory.
- When the Certificate Manager issues a new certificate (the request may originate from a Registration Manager), it publishes the certificate to the configured directory.
- When the Certificate Manager revokes a certificate (the request may originate from a Registration Manager), it removes that certificate from the configured directory.
- When a certificate expires, the Certificate Manager can remove that certificate from the configured directory. The server uses the Job Scheduler component to accomplish this task; see "Directory Update and Notification" on page 444).
- When the certificate revocation list is created or updated (either through the CMS window or through the certificate-revocation feature provided in the agent or end-entity interface), the Certificate Manager publishes that list to the configured directory.

To configure a Certificate Manager to publish certificates and CRLs to a directory, follow these steps:

- Step 1. Plan
- Step 2. Set Up the Directory for Publishing
- Step 3. Configure the Certificate Manager to Publish Certificates

- Step 4. Configure the Certificate Manager to Publish CRLs
- Step 5. Identify the Publishing Directory
- Step 6. Test Certificate and CRL Publishing (optional)

Step 1. Plan

Before configuring a Certificate Manager to publish its CA certificate, end-entity certificates, and CRLs to a directory, do this:

- Read "Publishing of Certificates to a Directory" on page 716 and "Publishing of CRLs to an LDAP Directory" on page 724 to understand how the Certificate Manager publishes certificates and CRLs to the directory.
- Read sections "Mapper Modules" on page 732 and "Publisher Modules" on page 749. Be sure to take a look at the default mappers and publishers created during CMS installation and determine whether they are suitable for your setup. If they're unsuitable, decide on the mapper and publisher modules you want to use.
- If you decided to not use the default mappers created using the `LdapCaSimpleMap` module, you will be required to manually create an entry for the CA in the publishing directory. (This document explains how to create an entry for the CA in Netscape Directory Server, version 4.x only.)
- Read "Publishing of CRLs" on page 721. Determine whether you want the Certificate Manager to publish version 1 or version 2 CRLs to the directory. If you decide to publish version 2 CRLs, read "CRL Extension Modules" on page 763 and determine the CRL extensions you want the Certificate Manager to set; you will be required to configure the server to set these extensions.
- Identify your publishing directory. If you've already configured the Certificate Manager to use an LDAP directory for authenticating users (for example, if you're using the directory-based or directory- and PIN-based authentication), you should consider publishing certificates and CRLs to the same directory. This way, users' security credentials will be kept with the rest of the user information.
- Note the following information for the directory: the host name, the port number, and the port type—whether it's an SSL or nonSSL port.

- Determine how you want the Certificate Manager to authenticate to the directory: whether to publish with basic authentication, publish over SSL without SSL client authentication, or publish over SSL with SSL client authentication. Accordingly, you will need to configure the Directory Server.

- If you want the Certificate Manager to authenticate to the directory using SSL client authentication, determine the certificate the Certificate Manager must use for SSL client authentication. By default, the server uses its SSL server certificate; see "SSL Server Key Pair and Certificate" on page 228.

  Depending on your PKI setup, you may use an external CA for requesting the certificate. For example, if your Certificate Manager is a subordinate CA to an external CA, you can get the Directory Server's certificate signed by the same CA that signed your Certificate Manager's certificate.

- If certificates the Directory Server and Certificate Manager will use during SSL-enabled communication already exist, check the CA that issued these certificates. The CA that issued the Directory Server's SSL server certificate must be trusted by the Certificate Manager. Similarly, the Directory Server must trust the CA that issued the certificate the Certificate Manager will use for client authentication.

- Determine how you want the Certificate Manager to bind to the directory: whether to bind as `CN=directory manager` or as another user; if it's another user, the entry must have read-write privileges to the directory tree that contains entries for end-entities to whom you intend to issue certificates.

- If you're not the directory administrator, consult the directory administrator about making changes to the schema, if required.

- Keep your directory documentation handy. For an online version of Netscape Directory Server 4.x documentation, check this file:

  `<server_root>/manual/index.html`

  You can find documentation for other versions of Netscape Directory Server at this site:

  `http://docs.iplanet.com/docs/manuals/directory.html`

Step 2. Set Up the Directory for Publishing

For a Certificate Manager to publish certificates and CRLs to an LDAP directory, the directory needs to be set up to receive certificate- and CRL-related information from the Certificate Manager.

- Step A. Verify the Directory Schema
- Step B. Add an Entry for the CA
- Step C. Identify an Entry That Has Write Access
- Step D. Verify Entries for End Entities
- Step E. Specify the Directory Authentication Method
- Step F. Modify the Certificate Mapping File
- Step G. Restart Directory Server

Step A. Verify the Directory Schema

For a Certificate Manager to publish certificates and CRLs to a directory, it must be configured with specific attributes and object classes. This section discusses those basic schema requirements. Is it assumed that you're familiar with directory schema and related terminology. If you're not, check the Directory Server documentation.

Required Schema for Publishing End-Entity Certificates

The Certificate Manager publishes an end entity's certificate to the `userCertificate;binary` attribute within the end entity's or subject's directory object. This attribute is multivalued; each value is a DER encoded binary X.509 certificate. The LDAP object class named `inetOrgPerson` allows this attribute. This object class is supported by Netscape Directory Server versions 1.0, 3.x, and 4.x. The mix-in object class named `strongAuthenticationUser` allows this attribute and can be combined with any other object class to allow certificate publication to that object. Note that the Certificate Manager does not automatically add this object class in the schema table of the corresponding Directory Server while publishing or

unpublishing end-entity certificates. If the directory object that it finds does not allow the `userCertificate;binary` attribute, the addition or removal of that specific certificate fails.

If you have created user entries as `inetOrgPerson`, the `userCertificate;binary` attribute already exists in the directory. Otherwise, you must add the `userCertificate;binary` attribute to your directory schema table. For information on modifying directory schema, check the Directory Server documentation.

Required Schema for Publishing the CA Certificate

The Certificate Manager publishes its own CA certificate in the `caCertificate;binary` attribute of the CA's directory object when the server is started; this is the object that corresponds to the Certificate Manager's issuer name. This is a required attribute of the `certificationAuthority` object class. Note that the Certificate Manager will add this object class to the directory entry for the CA, provided that it finds the CA's directory entry.

Required Schema for Publishing CRLs

The Certificate Manager maintains its list of revoked certificates in its internal database; this list is called the certificate revocation list (CRL). You can configure the server to publish the CRL to the directory whenever it is generated, which could be when a certificate is revoked and at regular intervals. You can also manually trigger the server to generate a CRL and publish it to the directory.

The Certificate Manager publishes the updated CRL to the CA's directory object under this attribute: `certificateRevocationList;binary`.

This attribute is an attribute of the object class `certificationAuthority`. The value of the attribute is the DER encoded binary X.509 certificate revocation list. The CA's entry must already be a certificate authority.

Step B. Add an Entry for the CA

Complete this step only if you want to manually create an entry for your CA in the directory—that is, you do not want use the automated feature built into the `LdapCaSimpleCAMap` plug-in module for creating the CA's entry in a directory; see "CA Certificate Mapper" on page 734.

For the Certificate Manager to publish its CA certificate and CRL, the directory must include an entry for the CA. This section explains how to manually add this entry in Netscape Directory Server 4.x using the Directory Server window (which you can launch from within Netscape Console). To add this entry in Netscape Directory Server 3.x, use its HTML forms-based interface (also called the HTTP gateway).

When adding the CA's entry to the directory, you need to select the entry type based on the distinguished name of your CA:

- If your CA's distinguished name begins with the `CN` component, create a new `person` entry for the CA. (If you select a different type of entry, the interface may not allow you to specify a value for the `CN` component.)
- If your CA's distinguished name begins with the `OU` component, create a new `organizational unit` entry for the CA.

After you select the correct entry type, you need to specify the required information to create the entry. Note that the entry you create doesn't have to be in the `certificationAuthority` object class. The Certificate Manager will convert this entry to the `certificationAuthority` object class automatically by publishing its CA's signing certificate (as explained in "Required Schema for Publishing the CA Certificate" on page 790).

To create an entry for the Certificate Manager in Netscape Directory Server, version 4.x:

1. Log in to Netscape Console (see "Logging In to Netscape Console" on page 69).
2. Locate the Directory Server instance you want the Certificate Manager to use for publishing certificates and CRLs.
3. Double-click the instance or select the instance and click Open.

   This opens the Directory Server window.
4. Select the Directory tab.
5. Select the domain name, right click, select New, and then select Other.

   The "New object" window appears.
6. Select "person" and click OK.

   The Property Editor - New window appears.

7. Enter the required information.

   Full name. Enter the common name (the value of the `CN` component) of the CA exactly as it appears in the issuer DN; this DN shows up in the CA's signing certificate. For example, if your CA's issuer DN is `CN=testCA, OU=Research Dept, O=Siroe Corporation, ST=California, C=US`, you should enter `testCA` in this field.

   Last name. Enter the name again; it must be the same as the one you entered in the "Full name" field.

8. Keep the default values in the remaining fields, and click OK.

   The new entry appears in the Directory tab.

9. Verify that the entry has been created.

   1. Double-click Directory Administrators, click Members, and then click Add.

   2. Search for the user entry you added earlier.

   3. Click OK and again OK.

Step C. Identify an Entry That Has Write Access

When you configure the Certificate Manager to work with Directory Server, you'll be required to specify a distinguished name in the directory that has read-write permissions to the directory. To publish certificates and CRLs to the directory, the Certificate Manager needs to use a user entry (in the directory) that has write access to the directory. This enables the Certificate Manager to bind to the directory as this user and modify the user entries with certificate-related information and the CA entry with CA's certificate and CRL related information.

To provide the Certificate Manager with a user entry that has read-write permission, you can do either of the following:

- Use the DN of an existing entry that has write access. For example, you can use the entry of the Directory Manager or choose an alternative.
- Give write access to the user entry you created for the Certificate Manager in the previous step. The entry can be identified by the Certificate Manager's DN. For example, it may look like this:

```
CN=testCA, OU=Research Dept, O=Siroe Corporation,
ST=California, C=US
```

For instructions on giving write access to the Certificate Manager's entry, see your LDAP directory documentation. In either case, note the entry DN and the corresponding password as you will be required to identify this user entry to the Certificate Manager later; see "Step 5. Identify the Publishing Directory" on page 830.

Step D. Verify Entries for End Entities

The publishing directory must contain an entry for each end entity for whom you want a certificate published. If the end entity does not have an entry in the directory, the Certificate Manager will not be able to publish the end entity's certificate.

To add an entry for each end entity, you can use the tools provided with Directory Server. Keep in mind that the end-entity entries must belong to an object class, such as `inetOrgPerson`, that allows the `userCertificate;binary` attribute.

Note If you configured the Certificate Manager to use directory-based authentication for end entities and are using the same directory for authentication and publishing, you may not have to deal with this issue. The server will not issue certificates to end entities that do not have entries in the directory. See "End-Entity Authentication During Certificate Enrollment" on page 265.

Step E. Specify the Directory Authentication Method

Depending on how you want the Certificate Manager to authenticate to the directory, you must set up Directory Server for one of the following methods of communication:

- Basic authentication
- SSL without client authentication
- SSL with client authentication

The instructions that follow explain how to configure Netscape Directory Server 4.x for all of the above methods of communication. If you're using any other directory, refer to the documentation that accompanied that product.

Publishing With Basic Authentication

To configure Directory Server for basic authentication:

1. Go to the Directory Server window.
2. Select the Configuration tab, and then in the right pane, select the Encryption tab.
3. Make sure that the Enable SSL box is unchecked. If it's checked, uncheck it.
4. Click Save.

   You are prompted to restart the server. Don't restart the server yet; you can do this after you've made all the configuration changes.

Publishing Over SSL Without Client Authentication

To configure the Directory Server for SSL-enabled communication:

1. Go to the Directory Server window.
2. Select the Configuration tab, and then in the right pane, select the Encryption tab.
3. Check the Enable SSL box.
4. In the Cipher Family section, check the RSA box.
5. Click the Cipher Preferences button and select the appropriate SSL ciphers.

   For details on individual ciphers, click the Help button.
6. In the Client Authentication section, select the "Allow client authentication" option.

   Be sure not to select the "Require client authentication" option. If you do, Netscape Console will not be able to communicate with the directory.
7. Click Save.

   You are be prompted to restart the server. Don't restart the server yet; you can do this after you've made all the configuration changes.

Publishing Over SSL With Client Authentication

For the Certificate Manager to publish to the directory with SSL client authentication, Directory Server must

- Contain an SSL server certificate in its certificate database
- Trust the CA that issued its SSL server certificate
- Trust the CA that issued the certificate the Certificate Manager will use for SSL client authentication
- Use a valid, secure port number for communication with the Certificate Manager
- Have SSL-enabled communication turned on in its configuration

The steps that follow explain how you can configure Directory Server for all of the above.

Step 1. Check the Directory Server's Certificate Database

Before getting an SSL server certificate, determine whether Directory Server already has an SSL server certificate installed in its certificate database and whether you want Directory Server to use the same certificate during the SSL handshake.

To check the Directory Server's certificate database:

1. Go to the Directory Server window.
2. Select the Tasks tab.
3. From the Console menu, choose the Manage Certificates option.

   The Certificate Management dialog box appears showing a list of all the certificates installed for Directory Server.
4. Scroll through the list to see if it contains the SSL server certificate that you want to use.

   - If the server has an SSL server certificate, check the CA that has issued the certificate. If this CA is trusted by the Certificate Manager, you can configure Directory Server to use the same certificate. If the CA is untrusted by the Certificate Manager and you want the Certificate

Manager to trust it, you need to check the Certificate Manager's certificate database for the CA certificate, add it if it isn't present, and specify that it be trusted. For instructions on manipulating the Certificate Manager's certificate database, see "Changing the Trust Settings of a CA Certificate" on page 299.

After you've made sure that the CA is trusted by the Certificate Manager, go to "Step 10. Verify the Port Number" on page 803.

- If the server does not have an SSL server certificate, or if you don't want the Certificate Manager to trust the CA that has issued the Directory Server's certificate, you must get an SSL server certificate for the Directory Server from a CA that is trusted by the Certificate Manager. You may get this certificate from the Certificate Manager itself.

  The instructions that follow (Step 2 through Step 9) explain how to do this.

Step 2. Generate an SSL Server Certificate Request for Directory Server

The steps below explain in general how to generate a certificate signing request (CSR) using the Certificate Setup Wizard, which is built into the Directory Server window available within Netscape Console. For detailed instructions on each step of the wizard, you should read the on-screen instructions and view the online help by clicking the Help button.

In the first step of generating the CSR, you will be asked to specify whether the certificate is for a new key pair or an exiting key pair and the method for submitting the CSR to the CA.

- If you want to request the certificate from an external CA, you should click the Show CA button to see whether the CA of your interest is listed there. If it is listed, you can open the SSL server enrollment interface of that CA so that you can paste the CSR the wizard will generate.
- If you want to request the certificate from the Certificate Manager, there are three possible ways in which you can submit the CSR to the Certificate Manager:

  — Submit the CSR directly from the wizard; in this method, you do not need to copy the CSR the wizard generates.

— Submit the CSR as an email to the CA's administrator; to use this method, you need to know the email address of the person who processes certificate requests for the CA and you need to copy the CSR the wizard generates.

— Submit the CSR manually by pasting it into the Certificate Manager's SSL server enrollment form; to use this method, you need to copy the CSR the wizard generates.

To generate a certificate request:

1. In the Directory Server window, select the Tasks tab, and then click the Certificate Setup Wizard button.

2. Select the token for generating the key pair (and for storing the certificate). Since you don't have the certificate, select No.

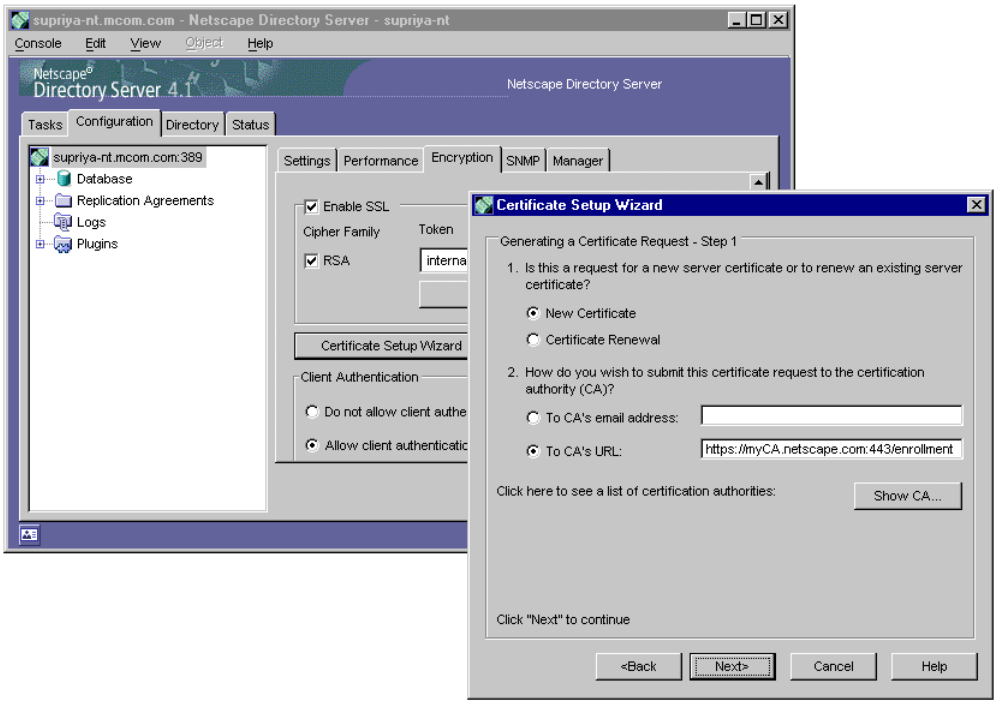
   If you're generating the certificate for the first time, the wizard informs you that it needs to create a trust database (`cert7.db` and `key3.db` files) for Directory Server.

3. When prompted for the password, enter the password.

   Remember this password because you will be required to supply it when starting Directory Server from now on.

   Once the trust database is generated, the wizard steps for generating the CSR begin.

4. You are asked to specify whether the certificate is for a new key pair or an existing key pair and the method for submitting the CSR to the CA.



The choices for submitting the CSR to the CA include the following:

To CA's email address. Select this if you want to send the CSR to the CA administrator's email address. Type the administrator's email address (for example, `jdoe@siroe.com`) in the adjoining field. The administrator will then be required to submit the request to the CA by pasting the CSR in the CA's server enrollment form.

To CA's URL. Select this if you want to submit the CSR to the Certificate Manager directly. Depending on the end-entity port that's enabled, type either of the following URL:

```
http://<CA's_host_name>:<end_entity_port>/enrollment
```

or

```
https://<CA's_host_name>:<end_entity_SSL_port>/enrollment
```

Note that the request submitted to the CA's URL gets queued for approval by the Certificate Manager agent.

5. When the wizard displays the CSR, if you are running the wizard on a Windows NT system, copy the CSR (displayed in its base-64 encoded format) to a text file. The information you copied should look similar to the sample shown below. Do not make any changes to it. (As indicated in the message, a copy of this information is also saved to the `/temp` file in the host machine's file system.)

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MMIIBnzCCAQgCAQAwXzELMAkGA1UEBhMCVXMxEzARBgNVBAgTCkNBTElGT1JOSUExHTAbBgNVBAoT
FpME5ldHNjYXBlIENvbW0uIENvcnAuMRwwGgYDVQQDExNzdXByaXlhLW50Lm1jb20uY29tMIGfMA0
GCSGSIb3DQEBAQUAA4GNADCBiQKBgQCk49jBib3SZQqTt5YtIGugnVOq38Y1CcB9xowCsapR+DIow
f8MUVWGRUT38mcX0lfpNT4hzW1aePiHersIMZFLxRgel0kEtJ0ClWfNQKzrnOfpL1H3CjcLjSM5hW
aFt0M5hfZEtPk+XBsMbu3dCtbRacxs0M2I0AVkf+kp24ePvqD8QIDAQABoAAwDQYJKoZIhvcNAQEE
BQADYEAdE0hZFHK6fAonMnHmNz46M96qqgtjwO3R9alt1l+0YWKslCjf+ThG38adNw1aH0qioW7yl
THJhpHF48M3SmmTqR7S3yKg+3ECLWbMYmUmd2wNTxsIdD4r8ySByxMIncVSvCjFOHJnifZCUqr+0N
ukZnqhDWqy0vqrGW71akstyyttdtttyyy790bfgfiwrytdnbjdgnhffnb0hgjyu08o=po=9=

-----END NEW CERTIFICATE REQUEST-----
```

Step 3. Submit the Request to a CA and Get the SSL Server Certificate

If you decided to submit the certificate request to an external CA, you need to go to that CA's enrollment area and use the form provided for requesting SSL server certificates. After you submit the request, hold on to the confirmation message until you receive the certificate. When the CA sends the certificate to you, complete the remaining configuration, starting from "Step 6. Install the Certificate in the Directory Server's Certificate Database" on page 801.

The instructions in this step explain how to request the SSL server certificate from the Certificate Manager manually. You should complete this step if you didn't use the auto-submit feature of the wizard to directly send the CSR to the Certificate Manager's URL.

To submit the request to the Certificate Manager manually:

1. Open a web browser window.
2. Go to the end-entity interface of the Certificate Manager (or to the Registration Manager that's connected to the Certificate Manager).
3. In the left frame, under Server, click SSL Server.

4. In the server enrollment form that appears, enter the required information:

   PKCS#10 Request. Paste the base-64 encoded blob, including the `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----` marker lines, you copied to the text file earlier.

   Name. Type your name.

   Email. Type your business email address, for example, `jdoe@siroe.com`.

   Phone. Type your business phone number.

   Additional Comments. Type any information that will help you identify this request in the future or will help the person who will process this request.

5. Click Submit.

Step 4. Approve the Request You Submitted

Skip to the next step if you submitted the CSR to an external CA. Complete this step if you submitted the CSR to the Certificate Manager.

To access the agent queue and approve the SSL server certificate request you submitted:

1. In the browser window, go to the Certificate Manager Agent Services interface. (If you submitted the request to a Registration Manager, go its agent interface.)

2. In the left frame, select List Requests. In the form that appears, select the "Show pending request" option and click Find.

3. In the request queue, locate the request you submitted and click Details.

4. Verify the information and click Do It.

   If your request contained all the required information, the server issues a certificate and you should see a message indicating so.

5. Click Show Certificate.

   The complete details about the certificate appear. Don't close the page; in the next step, you'll need to copy the certificate from this page.

Step 5. Copy the SSL Server Certificate

You must go through this step, irrespective of whether you submitted the CSR to the Certificate Manager or to an external CA.

To install the certificate in the Directory Server's database, you need to have a copy of the certificate in its base 64-encoded format:

- If you submitted the CSR to an external CA, wait till you receive the certificate. When you receive the certificate, look for the base 64-encoded blob of the certificate.
- If you submitted the CSR to the Certificate Manager, check the confirmation page that you received in the previous step; it contains the certificate in its base 64-encoded format.

The steps below explain how to copy the base 64-encoded blob of the certificate from the confirmation page that you received from the Certificate Manager:

1. In the page that shows the certificate details, scroll down to the section that says "Installing this certificate in a server".
2. Copy the base-64 encoded certificate, including the `----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file or to the clipboard; be sure not to make any changes to the text blob. An example of the information you should copy is shown below:

```
-----BEGIN CERTIFICATE-----
MMIICVDCCAf6gAwIBAgIBDDANBgkqhkiG9w0BAQQFADB6MQswCQYDVQQGEwJVUzELMAkGA1UECBMC
Q0M0ExFjAUBgNVBAcTDU1vdW50YWluIFZpZXcxETAPBgNVBAoTCE5ldHNjYXBlMRUwEwYDVQQLEwx
TZW1cml0eVB1YnMxHDAaBgNVBAMTE0NlcnRpZmljYXRlIE1hbmFnZXIwHhcNOTkwNzA5MjIxNjQ5W
hcMDAwNzA4MjIxNjQ5WjCQYDVQQGEwJVczETMBEGA1UECBMKQ0FMSUZPUk5JQTEdMBsGA1UEChMUT
m0c2NhcGUgQ29tbS4gQ29ycC4xHDAaBgNVBAMTE3N1cHJpeWEtbnQubWNvbS5jb20nbh4++oPxAgM
AAGjRjBEMBEGCWCGSAGG+EIBAQQEAwIGQDAOBgNVHQ8BAf8EBAMCBPAwHwYDVR0jBBgFoAUXuCIMl
07LCdanaxlek5DlpU4cxLgwDQYJKoZIhvcNAQEEBQADQQCagJsZwKG2usqiQ+bmZ0TJb44XhXDLRY
1GkbXtNLLf5acA1iBvi0cbEG5UsZk5zdB5zSDDe7Tuk9HrbyQrtQ6F
-----END CERTIFICATE-----
```

Step 6. Install the Certificate in the Directory Server's Certificate Database

You must go through this step, irrespective of whether you requested the certificate from the Certificate Manager or from an external CA.

To install the SSL server certificate in the Directory Server's certificate database:

1. Go to the Directory Server window.
2. Start the Certificate Setup Wizard.
3. In the first step that the wizard displays, select the "Install Certificate for This Server" option.
4. In the second step, select the "The certificate is located in the following text field" option and paste the certificate blob, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, you copied earlier.
5. Follow the prompts and add the certificate to the certificate database.

Step 7. Copy the CA Chain

You must go through this step, irrespective of whether you requested the certificate from the Certificate Manager or from an external CA.

The steps in this section explain how to copy the CA chain, if you requested the SSL server certificate from a Certificate Manager. If you got the certificate from an external CA, make sure that the CA's chain exists in certificate database of Directory Server; otherwise, go to the CA site and copy the chain.

1. Go to the end-entity interface of the Certificate Manager (or to the Registration Manager that's connected to the Certificate Manager).
2. Click the Retrieval tab.
3. In the left frame, click Import CA Certificate Chain.
4. In the form that appears, select the "Display the CA certificate chain in PKCS#7 for importing into a server" option, and click Submit.

   The CA certificate chain appears.
5. Copy the base-64 encoded certificate blob, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to a text file or the clipboard.

Step 8. Install the CA Chain in the Directory Server as a Trusted CA

To install the CA chain:

1. Go to the Directory Server window.
2. Start the Certificate Setup Wizard.
3. Select the option to install a certificate for a trusted certificate authority.
4. Select the "The certificate is located in the following text field" option and paste the certificate blob, including the `-----BEGIN CERTIFICATE----` and `-----END CERTIFICATE-----` marker lines, you copied earlier.
5. Follow the prompts and add the CA certificate chain to the certificate database of Directory Server.

Step 9. Confirm that the New Certificates are Installed

To verify that the certificates are installed in the certificate database of Directory Server:

1. In the Directory Server window, select the Tasks tab.
2. From the Console menu, select Manage Certificates.

   The Certificate Management dialog box appears showing a list of certificates installed for Directory Server.
3. Scroll through the list. You should find the certificates you installed. If you find the certificates, your server is ready for SSL activation.

Step 10. Verify the Port Number

Before turning on SSL-enabled communication for Directory Server, you must verify that the configured port number can be used for this purpose. If not, you must change the port number to a valid one.

To modify the port (for a secure port) on which the Directory Server listens for incoming requests:

1. In the Directory Server window, select the Configuration tab, and then in the navigation tree, select the root (the topmost) item.

2. Select the Settings tab in the right pane.

   Port. Type the port number you want the server to use for non-SSL communication. The default port number is 389.

   Encrypted Port. Type the port number you want the server to use for SSL-enabled communication. The default secure port number is 636. The encrypted port number that you specify must not be the same as one you specified in the Port field.

3. Click Save.

   You are be prompted to restart the server. Don't restart the server yet; you can restart it after you've made all the changes.

   Be aware that changing the Directory Server port number requires you to change the corresponding port number in all other servers that communicate with the directory. For example, if you have other Netscape Servers installed that point to the directory, you need to update those server configurations to use the new port number. For details, see *Managing Servers with Netscape Console* (to locate this document, see `<server_root>/manual/index.html`).

Step 11. Turn on SSL-Enabled Communication

To turn on SSL-enabled communication in Directory Server:

1. In the Directory Server window, select the Configuration tab, and then in the right pane, select the Encryption tab.

2. Check the Enable SSL box.

3. In the Cipher Family section, check the RSA box.

4. Click the Cipher Preferences button and select the appropriate SSL ciphers.

   For details on individual ciphers, click the Help button.

5. In the Token drop-down list, select the token that contains the key pair for the certificate you installed (or for the certificate you want the server to use).

6. Select the certificate you want the server to use during SSL-enabled communication with the Certificate Manager.

7. In the Client Authentication section, select the appropriate option:

   Do not allow client authentication. Select this if you want to configure the directory for basic authentication or for SSL-based communication without client authentication.

   Allow client authentication. Select this if you want to configure the directory for SSL client authenticated communication.

   Require client authentication. Don't select this option. If you do, Netscape Console will not be able to communicate with Directory Server. This is because Netscape Console does not support client-authenticated communication yet. For example, if you're using the same directory for user data and configuration information of other Netscape servers and if you configure Directory Server to require client authentication, you will no longer be able to manage your Netscape servers from Netscape Console; instead, you will be required to use the command-line tools.

8. Click Save.

   You are prompted to restart the server. Don't restart the server yet; you can restart it after you've made all the required changes.

Step F. Modify the Certificate Mapping File

This step explains how to modify the `certmap.conf` file to add a certificate mapping rule for the CA's entry you created. You need to go through this step only if you configured the directory for SSL client authenticated communication. Otherwise, skip to "Step G. Restart Directory Server" on page 810.

When the Certificate Manager presents its certificate for SSL client authentication, Directory Server uses the mapping rule specified in the `certmap.conf` file to locate the corresponding entry in the directory and then determine the access privileges set for the entry. The certificate mapping file is

located in the <server_root>/shared/config directory, where <server_root> is the directory in which the Directory Server binaries are installed.

The certmap.conf file specifies the following:

- Where in the directory tree the server should begin its search for locating the entry in the directory
- What certificate attributes the server should use as search criteria when searching for the entry in the directory
- Whether the server needs to go through any additional verification process

The file contains one or more named mappings, each applying to a different CA. A mapping has the following syntax:

```
certmap <name> <issuerDN>
<name>:<property1> [<value1>]
<name>:<property2> [<value2]
...
<name>:<propertyn> [<valuen]
```

The first line specifies a name for the entry and the DN of the issuer of the client certificate—in this case, the issuer of the certificate the Certificate Manager will present during client authentication. (By default, the Certificate Manager uses its SSL server certificate generated during installation.) The name is arbitrary; you can define it to be whatever you want. However, the issuer DN must exactly match the issuer DN of the CA that has issued the certificate the Certificate Manager will use for client authentication. For example, the following two issuer DN lines differ only in the number of spaces separating the attribute value assertions (AVAs), but the Directory Server will treat these two entries as different:

```
certmap moz CN=myCA,OU=myDept,O=myCompany,C=US
certmap moz CN=myCA,OU=myDept,O=myCompany, C=US
```

The second and subsequent lines in the named mapping match properties with values. The certmap.conf file has six default properties, but the ones that should be of use to you are explained below. For in depth detail about the certmap.conf file, see *Managing Servers with Netscape Console* (open the <server_root>/manual/index.html file to locate this document).

- `DNComps`—This is a list of comma-separated DN attribute tags used to determine where in the directory the server should start searching for directory entries that match the Certificate Manager's information (that is, the owner of the client certificate). The Directory Server gathers values for these tags from the certificate presented by the Certificate Manager during client authentication and uses the values to form an LDAP DN, which then determines where the server starts its search in the directory. For example, if you set `DNComps` to use the `<O=org>` and `<C=country>` DN attribute tags (`DNComps=O,C`) the server starts the search from the `O=<org>`, `C=<country>` entry in the directory, where `<org>` and `<country>` are replaced with values from the values specified in the subject DN of the certificate presented for client authentication.

  — If the `DNComps` entry is present but has no value, the server searches the entire LDAP tree for entries matching the filter.

  — If there isn't a `DNComps` entry in the mapping, the server uses either the `CmapLdapAttr` setting (if present) or the entire subject DN in the Certificate Manager's certificate.

  The following component tags are supported for `DNComps`: `CN`, `OU`, `O`, `C`, `L`, `ST`, `E`, and `Mail`. Case is ignored. You can use `E` or `Mail`, but not both.

- `FilterComp`—This is a list of comma-separated DN attribute tags used to create a filter by gathering information from the subject DN in the certificate presented during client authentication. Directory Server uses the values for these tags to form the search criteria for matching entries in the directory. If Directory Server finds one or more entries in the directory that match the Certificate Manager's information gathered from the certificate, the search is successful and the server optionally performs a verification. For example, if `FilterComps` is set to use the attribute tags `E` and `UID` (`FilterComps=E,UID`), the server searches the directory for an entry whose values for `E` and `UID` match the Certificate Manager's information gathered from the client certificate. Email addresses and user IDs are good filters because they are usually unique entries in the directory.

  Note that the filter needs to be specific enough to match only the Certificate Manager's entry in the LDAP directory. The following component tags are supported for `FilterComps`: `CN`, `OU`, `O`, `C`, `L`, `ST`, `E`, and `Mail`. Case is ignored. You can use `E` or `Mail`, but not both.

- `verifycert`—This tells the server whether it should compare the certificate the Certificate Manager presents during client authentication with the certificate found in the Certificate Manager's entry in the directory. It takes

one of the two values: `on` or `off`. It is recommended that you set this to `on` for a complete single sign-on solution. This ensures that Directory Server will authenticate the Certificate Manager unless the certificate presented exactly matches the certificate stored in the directory.

The following two examples illustrate two different ways you can use the `certmap.conf` file.

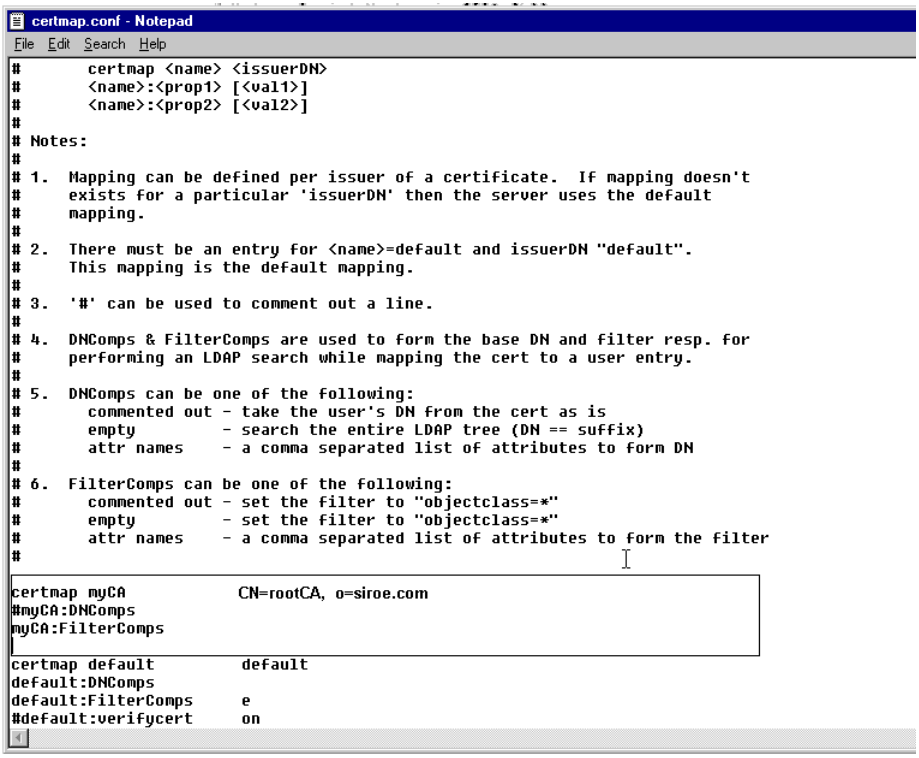
```
certmap default default
default:dnComps
default:filterComps E, UID

certmap MyCA CN=CA,OU=MyGroup,O=MyCompany,C=US
MyCA:dnComps OU,O,C
MyCA:filterComps E
MyCA:verifycert on
```

This file has two mappings: a default one and another for `MyCA`. When the Directory Server gets a certificate from anyone other than `MyCA`, the server uses the default mapping, which starts at the top of the LDAP tree and searches for an entry matching the client's email address and user ID. If the certificate is from `MyCA`, the server starts its search at the LDAP branch containing the organizational unit and searches for matching email addresses. Also note that if the certificate is from `MyCA`, the server verifies the certificate with the one stored for the entry in the directory; other certificates are not verified. Note that the issuer DN in the certificate must be identical to the issuer DN listed in the first line of the mapping. Even an extra space after a comma will cause a mismatch.

To modify the `certmap.conf` file:

1. In the Directory Server host machine, go to this directory: `<server_root>/shared/config`
2. Open the `certmap.conf` file in a text editor.

3. Follow the instructions in the file and add the mapping information for the entry you added.



The figure above shows the following mapping rule being added to the file:

```
certmap myCA            CN=rootCA, O=siroe.com
#myCA:DNComps
myCA:FilterComps
```

This mapping rule specifies that if the name of the CA that signed the certificate used for SSL client authentication by the Certificate Manager is `myCA` and that the issuer name or DN of the CA is `CN=rootCA, O=siroe.com`, the server should use the `FilterComps` attributes to locate the entry.

If you determine that the `certmap.conf` file needs an empty `DNComps` mapping (because your certificate subject name has no overlap with the corresponding directory DN), you may need to modify the default base DN in Directory Server by adding the following to the Directory Server configuration file:

```
dn: cn=config
changetype: modify
replace: nsslapd-certmap-basedn
nsslapd-certmap-basedn: dc=siroe, dc=com
```

4. Save your changes, and close the file.

Step G. Restart Directory Server

For all your changes to take effect, you must restart Directory Server.

- Starting Directory Server

  If you configured the Directory Server for basic authentication or SSL-enabled communication without client authentication, you can start the server from the Directory Server window from within Netscape Console:

  1. Click the Tasks tab.
  2. Click Restart the Server.

- Starting SSL-Enabled Directory Server

  If you configured the Directory Server for SSL-enabled communication with client authentication, here's how you must start the server:

  — On Windows NT, start the server from the server's host machine and supply the PIN or password that protects the key pair you generated for the Directory Server's certificate. For security reasons, the dialog box that prompts you for this PIN appears only on the server's host machine.

  — On Unix, start the server from the command line.

Step 3. Configure the Certificate Manager to Publish Certificates

This section explains how to specify certificate mapping and publishing rules the Certificate Manager should use to publish certificates to the correct entries in the directory.

- Step A. Modify the Default Mappers, Publishers, and Publishing Rules

- Step B. Add Mappers, Publishers, and Publishing Rules

Step A. Modify the Default Mappers, Publishers, and Publishing Rules

Complete this step if you decided to use any of the default mappers, publishers, and publishing rules created during installation. If you want to create new mappers, publishers, and publishing rules, skip to the next step, "Step B. Add Mappers, Publishers, and Publishing Rules" on page 816.

During installation, the Certificate Manager automatically creates a set of mappers that you would most likely want to use. The names of the default mappers are as follows:

- `LdapUserCertMap` (see "LdapUserCertMap Mapper" on page 747)
- `LdapCrlMap` (see "LdapCrlMap Mapper" on page 738)
- `LdapCaCertMap` (see "LdapCaCertMap Mapper" on page 737)

Similar to mappers, the Certificate Manager also creates a set of publishers for your convenience. The names of the default publishers are as follows:

- `LdapCaCertPublisher` (see "LdapCaCertPublisher Publisher" on page 755)
- `LdapCrlPublisher` (see "LdapCrlPublisher Publisher" on page 759)
- `LdapUserCertPublisher` (see "LdapUserCertPublisher Publisher" on page 757)

The Certificate Manager also creates a set of publishing rules using the default mappers and publishers. The names of these rules are as follows:

- `LdapCrlRule`
- `LdapCaCertRule`
- `LdapUserCertRule`

It is important that you review each of the default mappers, publishers, and publishing rules and modify them as suitable. The instructions below explain how to modify the default mappers, publishers, and publishing rules.

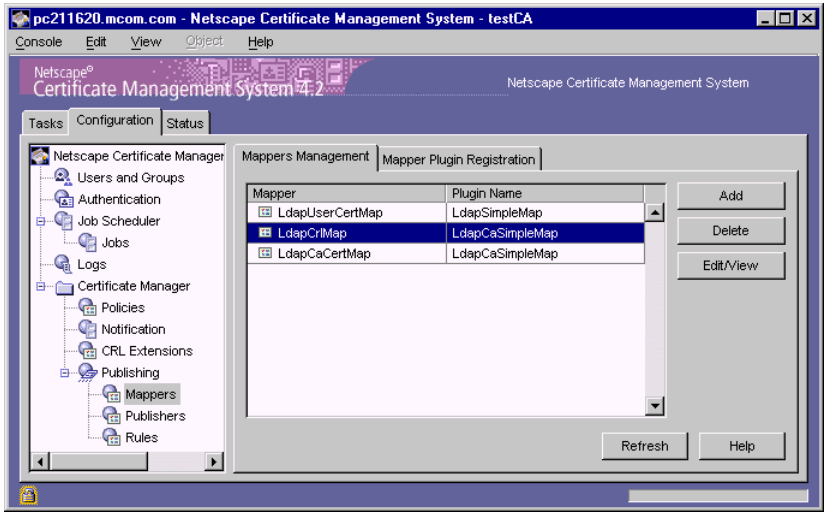
Step A.1. Modify the Default Mappers

You can modify a mapper by editing its configuration parameter values; you cannot change the name of a mapper. To change the name of a mapper, you need to create a new mapper exactly like the mapper you want to rename, except with a new name, and delete the old mapper.

To modify a mapper:

1. Log in to the CMS window for the Certificate Manager (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Publishing, and then select Mappers.

   The right pane shows the Mappers Management tab, which lists configured mappers.

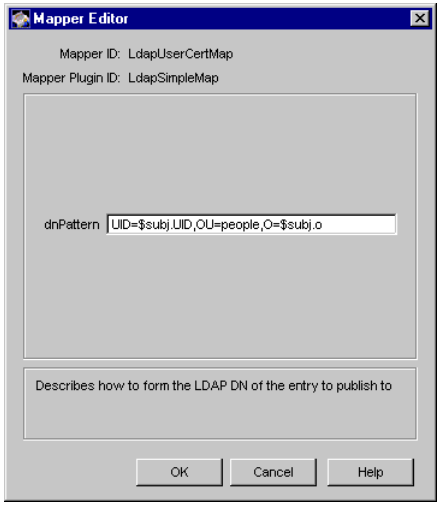


4. In the Mapper list, select a mapper that you want to modify.

   For the purposes of completing this instruction, assume that you selected the mapper named `LdapUserCertMap`.

5. Click Edit/View.

The Mapper Editor window appears, showing how this mapper is configured. An example is shown below.



6. Make the necessary changes and click OK.

   You are returned to the Mappers Management tab.

7. To modify the remaining mappers, repeat steps 4 through 6.

8. Click Refresh to see the update status of all the mappers.

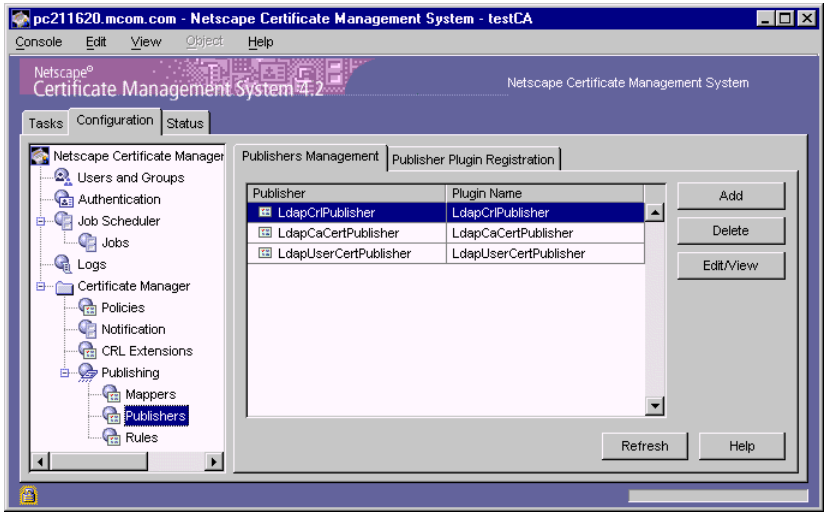
Step A.2. Modify the Default Publishers

Modifying a publisher involves changing its configuration parameter values; you cannot change the name of a publisher. To change the name of a publisher, create a new publisher using the same publisher plug-in module with the same parameter values, and delete the old one.

To modify a publisher:

1. In the navigation tree, select Publishing, and then select Publishers.

   The right pane shows the Publishers Management tab, which lists configured publishers.



2. In the Publisher list, select a publisher that you want to modify.

   For the purposes of this instruction, assume that you selected the publisher named `LdapUserCertPublisher`.

3. Click Edit/View.

   The Publisher Editor window appears, showing how this publisher is currently configured.

4. Make the necessary changes and click OK.

   You are returned to the Publishers Management tab.

5. To modify the remaining publishers, repeat steps 2 through 4.

6. Click Refresh to see the update status of all the publishers.

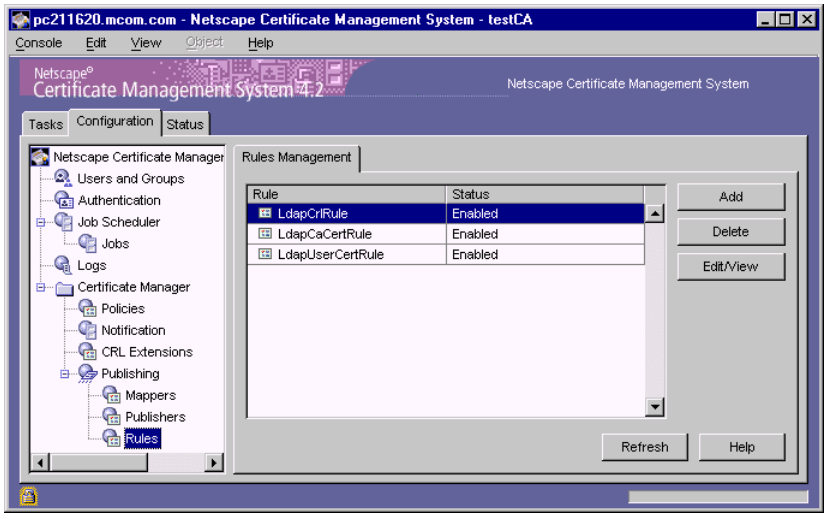
Step A.3. Modify the Default Publishing Rules

Modifying a publishing rule involves changing its configuration parameter values; you cannot change the name of a publishing rule. To change the name of a publishing rule, create a new rule with the same parameter values, and delete the old one.

To modify a publishing rule:

1. In the navigation tree, select Publishing, and then select Rules.

   The right pane shows the Rules Management tab, which lists configured publishing rules.



2. In the Rule list, select a publishing rule that you want to modify.

   For the purposes of this instruction, assume that you selected the rule named `LdapUserCertRule`.

3. Click Edit/View.

   The Rule Editor window appears, showing how the rule is currently configured.

4. Make the necessary changes and click OK.

   You are returned to the Rules Management tab.

5. To modify the remaining rules, repeat steps 2 through 4.

6. Click Refresh to see the update status of all the rules.

Step B. Add Mappers, Publishers, and Publishing Rules

Complete this step if you need to create new mappers, publishers, or publishing rules. For example, if you already configured the Certificate Manager for publishing all types of certificates in "Step A. Modify the Default Mappers, Publishers, and Publishing Rules" on page 811, you can skip to the next step, "Step A. Specify CRL Details" on page 824.

The instructions that follow cover how to add new mappers, publishers, and publishing rules for a CA certificate and for end-entity certificates. Creating of new mappers, publishers, and publishing rules for CRLs is covered in "Step 4. Configure the Certificate Manager to Publish CRLs" on page 823.

Follow the steps that's appropriate for you:

- Step B.1. Create a Mapper for the CA Certificate
- Step B.2. Create a Mapper for End-Entity Certificates
- Step B.3. Create a Publisher for the CA Certificate
- Step B.4. Create a Publisher for End-Entity Certificates
- Step B.5. Create a Publishing Rule for the CA Certificate
- Step B.6. Create Publishing Rules for End-Entity Certificates

Step B.1. Create a Mapper for the CA Certificate

Creating a mapper for the CA certificate involves creating an instance of the mapper module that enables the Certificate Manager to locate the CA's entry in the directory; for a list of modules, see "Overview of Mapper Modules" on page 732. Later, when creating the LDAP publishing rule for the CA certificate, you specify the mapper you create here.

To create a mapper:

1. In the navigation tree of the CMS window, under Publishing, select Mappers.

   The right pane shows the Mappers Management tab, which lists configured mappers.

2. Click Add.

   The Select Mapper Plugin Implementation window appears. It lists registered mapper modules.

3. Select a module.

   The following choices are the ones provided by default with the Certificate Manager for mapping a CA's certificate to the CA's directory entry. (If you have registered any custom mapper modules, they too will be available here for selection.)

   LdapDNCompsMap. Select this if you want the server to locate the CA's entry by formulating the entry's distinguished name from components in the certificate subject name and using it as the search DN. For details, see "DN Components Mapper" on page 738.

   LdapDNExactMap. Select this if you want the server to locate the CA's entry by searching for its certificate subject name. For details, see "Subject Name Mapper" on page 744.

   LdapSimpleMap. Select this if you want the server to locate the CA's entry by formulating the entry's DN from components specified in the certificate subject name and attribute variable assertion (AVA) constants. For details, see "Simple Mapper" on page 745.

   LdapSubjAttrMap. Select this if you want the server to locate the CA's entry by searching for an LDAP attribute whose value matches the certificate subject name. For details, see "Subject Attribute Mapper" on page 747.

   For the purposes of this instruction, assume that you selected `LdapDNCompsMapper`.

4. Click Next.

   The Mapper Editor window appears.

5. Enter the appropriate information:

   Mapper ID. Type a unique name for the mapper that will help you identify it; use an alphanumeric string with no spaces.

   baseDN. Type the DN from which the server should start searching for the CA's entry in the directory. If you leave the next field, `dnComps`, blank, the server uses the base DN value to start its search in the directory. For example, `O=siroe.com`.

   dnComps. Type DN components (attributes) separated by commas, that you want the server to use to locate an LDAP entry that match the CA's information. The server gathers values for these attributes from the CA certificate subject name and uses the values to form an LDAP DN, which then determines where in the LDAP directory the server starts its search. For example, if the subject name of your CA's certificate is `CN=testCA, O=siroe.com, C=US`, and you set `dnComps` to use the `O` and `C` attributes of the DN, the server starts the search from the `O=siroe.com`, `C=US` entry in the directory.

   If you leave the `dnComps` field empty, the server checks the value in the `baseDN` field and searches the directory tree specified by that DN. The server searches the entire LDAP tree for entries matching the filter specified by `filterComps` parameter values.

   filterComps. Type components the server should use to filter entries that result from the search. The server uses the `filterComps` values to form an LDAP search filter for the subtree. The server constructs the filter by gathering values for these attributes from the certificate subject name; it uses the filter to search for and match entries in the LDAP directory.

   If you need additional details about any of these parameters, click the Help button.

6. Click OK.

   The Mappers Management tab appears, listing the new mapper.

Step B.2. Create a Mapper for End-Entity Certificates

Creating a mapper for end-entity certificates involves creating an instance of the mapper module that enables the Certificate Manager to locate the correct end-entity entry in the directory; for a list of modules, see "Overview of Mapper Modules" on page 732. Later, when creating the publishing rule for end-entity certificates, you specify the mapper you create here.

To create a mapper for end-entity certificates, follow the procedure in Step B.1, above. Unlike the CA certificate mapper configuration, keep this mapper's configuration generic so that the Certificate Manager is able to locate any end-entity entry in the directory.

Step B.3. Create a Publisher for the CA Certificate

Creating a publisher for the CA certificate involves creating an instance of the publisher module that enables the Certificate Manager to publish the CA certificate to the correct attribute in the CA's directory entry; for a list of modules, see "Overview of Publisher Modules" on page 749. Later, when creating the LDAP publishing rule for the CA certificate, you specify the publisher you create here.

To create a publisher:

1. In the navigation tree of the CMS window, under Publishing, select Publishers.

   The right pane shows the Publishers Management tab, which lists configured publishers.

2. Click Add.

   The Select Publisher Plugin Implementation window appears. It lists registered publisher modules.

3. Select the module named `LdapCaCertPublisher`.

   As explained in "CA Certificate Publisher" on page 753, only this module publishes the CA certificate to `caCertificate;binary` attribute in the CA's directory entry. (If you have registered any custom publisher modules, they too will be available here for selection.)

4. Click Next.

   The Publisher Editor window appears.

5. Enter the appropriate information:

   Publisher ID. Type a unique name for the publisher that will help you identify it later; be sure to use an alphanumeric string with no spaces.

   caCertAttr. The field shows `caCertificate;binary`, the directory attribute to publish the CA certificate. Leave it as it is. If the field is empty, type `caCertificate;binary`.

caObjectClass. The field shows `certificationAuthority`, the object class for the CA's entry in the directory. Leave it as it is. If the field is empty, type `certificationAuthority`.

6. Click OK.

   The Publishers Management tab appears, listing the new publisher.

Step B.4. Create a Publisher for End-Entity Certificates

Creating a publisher for end-entity certificates involves creating an instance of a publisher module that enables the Certificate Manager to publish an end-entity certificate to the correct attribute in the end entity's directory entry; for a list of modules, see "Overview of Publisher Modules" on page 749. Later, when creating publishing rules for end-entity certificates, you specify the publisher you create here.

To create a publisher for end-entity certificates, complete the procedure in Step B.3 above. When selecting the publisher module, be sure to choose the module named `LdapUserCertPublisher` as this is the only module that allows publishing to the `userCertificate;binary` attribute of a mapped-directory entry.

Step B.5. Create a Publishing Rule for the CA Certificate

Creating a publishing rule for the CA certificate involves creating a rule that uses the mapper and publisher that you created for the CA certificate in the previous steps.

To create a publishing rule:

1. In the navigation tree, under Publishing, select Rules.

   The right pane shows the Rules Management tab, which lists configured publishing rules.

2. Click Add.

   The Select Rule Plugin Implementation window appears. It lists registered modules that enable creating of publishing rules.

3. Select the module named `Rule`.

   This is the default module. (If you have registered any custom modules, they too will be available for selection.)

4. Click Next.

   The Rule Editor window appears.

5. Enter the appropriate information:

   Rule ID. Type a unique name for the rule; use an alphanumeric string with no spaces.

   enable. Select this option.

   predicate. Type `HTTP_PARAMS.certType==ca`, indicating that the rule be applied to the CA certificate only. (For information on predicates, see "Using Predicates in Policy Rules" on page 490.)

   type. Select `cacert`.

   mapper. Select the mapper you added for locating the CA's entry in the directory.

   publisher. Select the publisher you added for publishing the CA's certificate to the directory.

6. Click OK.

   The Rules Management tab appears, listing the new rule.

Step B.6. Create Publishing Rules for End-Entity Certificates

Creating a publishing rule for end-entity certificates involves creating a rule for publishing each type of end-entity certificates the Certificate Manager will issue:

- SSL client certificates
- SSL server certificates
- Object signing certificates
- Registration Manager signing certificates
- OCSP responder certificates
- Router certificates

You need to create a rule for each type of certificate using the mapper and publisher that you created for end-entity certificates.

To create a publishing rule:

1. In the navigation tree, under Publishing, select Rules.

   The right pane shows the Rules Management tab, which lists configured publishing rules.

2. Click Add.

   The Select Rule Plugin Implementation window appears. It lists registered modules that enable creating of publishing rules.

3. Select the module named `Rule`.

   This is the default module. (If you have registered any custom modules, they too will be available for selection.)

4. Click Next.

   The Rule Editor window appears.

5. Enter the appropriate information:

   Rule ID. Type a name for the rule; use an alphanumeric string with no spaces.

   enable. Select this option.

   predicate. Type `HTTP_PARAMS.certType==client`, indicating that the rule be applied to client certificates only (see Table 22.1).

   type. Select `certs`.

   mapper. Select the mapper you added for locating end-entity entries in the directory.

   publisher. Select the publisher you added for publishing end-entity certificates (to the `userCertificate;binary` attribute of an end-entity entry in the directory).

6. Click OK.

   The Rules Management tab appears, listing the new rule you just created for publishing end users' client certificates.

7. Repeat steps 1 through 6 for each type of end-entity certificate the Certificate Manager will issue. Use Table 22.1 for filling in the correct values in the `type` and `predicate` fields. (For information on predicates, see "Using Predicates in Policy Rules" on page 490.)

Table 22.1 Certificate type and predicate expression

End-entity certificate type	"type" field value	"predicate" field value
SSL client certificate	`certs`	`HTTP_PARAMS.certType==client`
SSL server certificate	`certs`	`HTTP_PARAMS.certType==server`
Object signing certificate	`certs`	`HTTP_PARAMS.certType==objSignClient`
Certificate Manager signing certificate (subordinate CA)	`cacert`	`HTTP_PARAMS.certType==ca`
Registration Manager signing certificate	`certs`	`HTTP_PARAMS.certType==ra`
OCSP responder certificate	`certs`	`HTTP_PARAMS.certType==ocspResponder`
Router certificate	`certs`	`HTTP_PARAMS.certType==CEP-Router`

Step 4. Configure the Certificate Manager to Publish CRLs

If you don't want the Certificate Manager to publish CRLs to the directory, skip to "Step 5. Identify the Publishing Directory" on page 830.

You can configure the Certificate Manager to publish CRLs to the directory that is currently configured for publishing the CA and end-entity certificates. A configured Certificate Manager will publish the CRL to the CA's entry in the specified directory, replacing the old CRL with the new one; the old CRL is not saved. The Certificate Manager connects to the directory using the base DN and password that you will specify in "Step 5. Identify the Publishing Directory" on page 830.

To configure a Certificate Manager to publish CRLs to the directory, follow these steps:

- Step A. Specify CRL Details

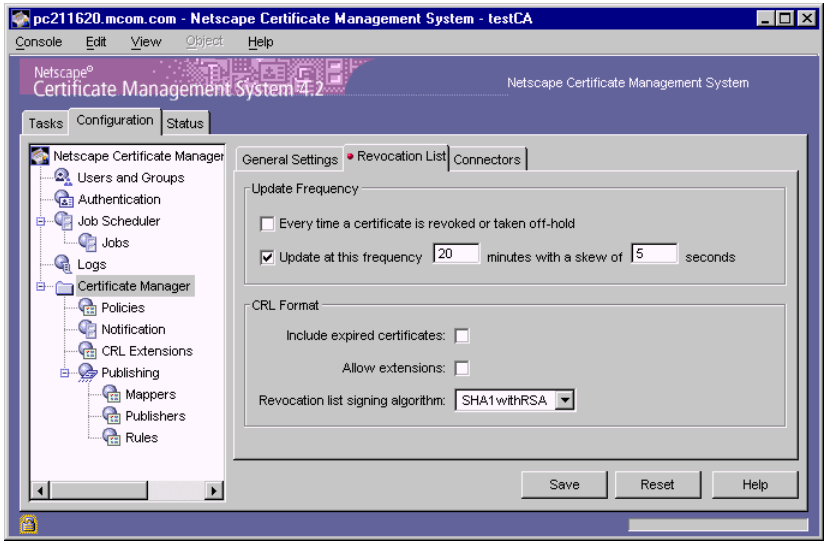
- Step B. Set the CRL Extensions
- Step C. Create a Mapper for the CRL
- Step D. Create a Publisher for the CRL
- Step E. Create a Publishing Rule for the CRL

Step A. Specify CRL Details

You can specify information, such as the publishing interval, the CRL version (whether to include CRL extensions), and the signing algorithm the Certificate Manager should use for signing the CRL object.

To specify CRL details:

1. In the navigation tree of the CMS window, select Certificate Manager, and then in the right pane, select the Revocation List tab.



2. In the Update Frequency section, specify the interval for publishing the CRL to the directory:

   Every time a certificate is revoked, or taken off-hold. Select this option if you want the Certificate Manager to generate the CRL every time it revokes a certificate. Keep in mind that the Certificate Manager attempts to

publish the CRL to the configured directory whenever the CRL is generated, in this case, every time a certificate is revoked. Publishing a CRL can be time consuming if the CRL is large. Configuring the Certificate Manager to publish CRLs every time a certificate is revoked may engage the server for a considerable amount of time; during this time, the server will not be able to service any requests it receives and will not be able to update the directory with any changes it receives.

Update at this frequency. Select this option if you want the Certificate Manager to generate CRLs at regular intervals. In this case, the server publishes the CRL to the configured directory at the interval you specify.

In the adjoining text field, type the interval, in minutes, at which the Certificate Manager should publish CRLs. For example, if you want the server to publish CRLs every day, you should type 1440 in this field.

with a skew of. If you configure the server to update the CRL automatically every time period, the server by default adds a 5 second skew to the next update time to allow time to create the CRL and publish it. For example, if you configure the server to update the CRL every 20 minutes, and if the CRL is updated at 16:00:00, the CRL will be updated again at 16:19:55. You can configure the skew by changing the default value, which is specified in seconds.

3. In the CRL Format section, specify the format for publishing the CRL:

   Include expired certificates. Check this box if you want the server to include revoked certificates that have expired in the CRL.

   Allow extensions. Check this box if you want to allow extensions in the CRL. If you enable this option, the server generates and publishes CRLs conforming to X.509 version 2 standard. If you disable this option, the server generates and publishes CRLs conforming to X.509 version 1 standard. By default, the server publishes version 1 CRLs. If you enable this option, be sure to set the required CRL extensions as described in "Step B. Set the CRL Extensions" on page 826.

   Revocation list signing algorithm. Select the algorithm the server should use to sign the CRL. If the Certificate Manager's signing key type is RSA, select `MD2 with RSA`, `MD5 with RSA`, or `SHA-1 with RSA`. If the Certificate Manager's signing key type is DSA, select `SHA-1 with DSA`.

4. To save your changes, click Save.

If the changes you made require you to restart the server, you are prompted accordingly. However, don't restart the server yet; you can restart it after you've made all the required changes.

Step B. Set the CRL Extensions

Complete this step only if you configured the Certificate Manager to publish version 2 CRLs—that is, you selected the "Allow extensions" option in "Step A. Specify CRL Details" on page 824.

During installation, the Certificate Manager creates default CRL extension rules; these are listed in Table 21.10 on page 766. Note that the server is configured to add the CRL Reason extension only; all the other rules are in the disabled state. In this step, you modify the default CRL extension rules to add the required CRL extensions.

To specify the CRL extensions the Certificate Manager should set:

1. In the navigation tree, under Certificate Manager, select CRL Extensions.

   The right pane shows the CRL Extensions Management tab, which lists configured extensions.

2. To modify a rule, select it and then click Edit/View.

3. Change the information as appropriate.

   Be sure to supply all the required values. Click the Help button for detailed information on individual parameters.

4. Click OK.

   You are returned to the CRL Extensions Management tab.

5. To modify other rules, repeat steps 2 through 4.

6. Click Refresh to see the updated status of all the rules.

Step C. Create a Mapper for the CRL

The Certificate Manager publishes the CRL to the `certificateRevocationList;binary` attribute of the CA's directory entry. (See "Required Schema for Publishing CRLs" on page 790.)

Since you already created a mapper for locating the CA's entry (either in "Step A. Modify the Default Mappers, Publishers, and Publishing Rules" on page 811 or in "Step B.1. Create a Mapper for the CA Certificate" on page 816), you can configure the Certificate Manager to use that mapper to locate the CA's entry for publishing the CRL; you don't need to create another mapper for publishing CRLs.

Step D. Create a Publisher for the CRL

Creating a publisher for the CRL involves creating an instance of the publisher module that enables the Certificate Manager to publish the CRL to the correct attribute in the CA's directory entry. In the next step, described in "Step E. Create a Publishing Rule for the CRL" on page 829, you specify the publisher you create here.

To create a publisher for the CRL:

1. In the navigation tree, click Publishers.

   The right pane shows the Publishers Management tab, which lists configured publisher instances.

2. Click Add.

   The Select Publisher Plugin Implementation window appears. It lists registered publisher modules.

3. Select the module named `LdapCrlPublisher`.

   Only this publisher module enables the Certificate Manager to publish the CRL to the `certificateRevocationList;binary` attribute of the CA's directory entry; for details, see "CRL Publisher" on page 757. (If you have registered any custom publisher modules, they too will be available for selection.)

4. Click Next.

   The Publisher Editor window appears.

5. Enter the appropriate information:

   Publisher ID. Type a name for the rule; use an alphanumeric string with no spaces.

   crlAttr. Make sure this field shows the directory attribute to publish the CRL, `certificateRevocationList;binary`. If necessary, type it in.

6. Click OK.

   The Publishers Management tab appears, listing the new publisher.

Step E. Create a Publishing Rule for the CRL

Creating a publishing rule for the CRL involves creating a rule that uses the mapper and publisher instances that you created in the previous steps. To create a publishing rule:

1. In the navigation tree, click Rules.

   The right pane shows the Rules Management tab, which lists any currently configured publishing rules.

2. Click Add.

   The Select Rule Plugin Implementation window appears. It lists registered modules that enable creating of publishing rules.

3. Select the module named `Rule`.

   This is the default module. (If you have registered any custom modules, they too will be available for selection.)

4. Click Next.

   The Rule Editor window appears.

5. Enter the appropriate information:

   Rule ID. Type a name for the rule; be sure to use an alphanumeric string with no spaces.

   enable. Select this option.

   predicate. Leave this field blank.

   type. Select `crl`.

   mapper. Select the mapper you added for locating the CA's entry in the directory.

   publisher. Select the publisher you added for publishing the CRL.

6. Click OK.

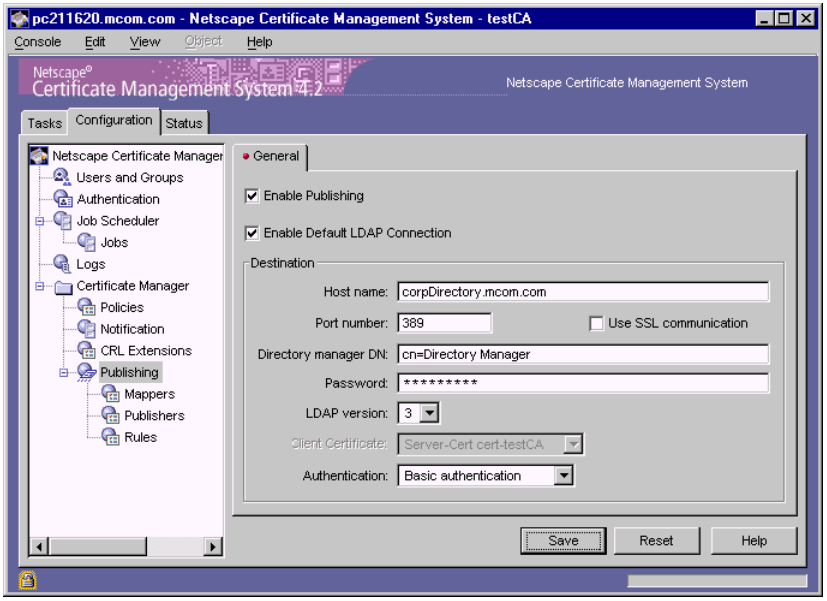
   The Rules Management tab appears, listing the new rule.

Step 5. Identify the Publishing Directory

To identify the directory to which the Certificate Manager should publish the CA certificate, end-entity certificates, and CRLs:

1. In the navigation tree of the CMS window, select Certificate Manager, and then select Publishing.

   The right pane shows the publishing details necessary for the server to publish to an LDAP-compliant directory.



2. To enable LDAP publishing, select both “Enable Publishing” and “Enable default LDAP connection” options.

3. In the Destination section, identify the Directory Server instance.

   Host name. Type the full host name of the Directory Server instance in this format: `<machine_name>.<your_domain>.<domain>`

   The Certificate Manager uses this name to locate the directory.

   If you configured the Directory Server for SSL client authenticated communication (in “Step E. Specify the Directory Authentication Method” on page 793), the name you enter here must match the `CN` component in the subject DN of the Directory server’s SSL server certificate. For example, the host name may look like `corpDirectory.siroe.com`.

   Port number. Type the TCP/IP port number at which the Directory Server is listening to certificate and CRL publishing requests from the Certificate Manager; you specified this port in “Step 10. Verify the Port Number” on page 803. The port you specify must be unique on the Directory Server host system; make sure no other application is attempting to use the port.

Authentication. Select the authentication type appropriate to your Directory Server configuration. The choices are `Basic authentication` and `SSL client authentication`.

If you configured the Directory Server for basic authentication or for SSL communication without client authentication, select `Basic authentication` and specify values for the Directory manager DN and password.

If you configured the Directory Server for SSL communication with client authentication, select `SSL client authentication`, select the `Use SSL communication` option, and identify the certificate that the Certificate Manager must use for SSL client authentication to the directory.

Use SSL communication. Select this option if the port number you specified is an SSL port; deselect the box if the port is non-SSL. The type of port you specify determines whether the Certificate Manager needs to do SSL client authentication prior to publishing certificates and CRLs to the directory.

Client certificate. Select the certificate you want the Certificate Manager to use for SSL client authentication to the publishing directory. By default, the Certificate Manager uses its SSL server certificate for this purpose (see "SSL Server Key Pair and Certificate" on page 228).

Directory manager DN. Type the distinguished name (DN) of the directory entry that you identified in "Step C. Identify an Entry That Has Write Access" on page 792. The Certificate Manager uses this DN to access the directory tree and to publish to the directory. The access control set up for this DN determines whether the Certificate Manager can perform publishing. Typically, you would want to enter the directory manager's DN because it has read-write permission to the entire directory tree (the root DN). For more information on root DN, see "Root Distinguished Name" on page 1155.

Password. Type the password for this DN. The Certificate Manager saves this password in the single sign-on password cache and uses it during startup; for details, see "Required Start-up Information" on page 128. (If you change the password, the server updates the single sign-on password cache with the new password.)

LDAP version. Select the version of LDAP protocol appropriate to your version of Directory Server. If the directory you want the Certificate Manager to publish to is based on Netscape Directory Server 1.x, select version 2. For Directory Server versions 3.x and later, select LDAP version 3.

4. To save your changes, click Save.

   The server attempts to connect to the specified Directory Server. If the information you specified is incorrect, the server displays an error message and you will need to correct the information and save your changes again.

   If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Step 6. Test Certificate and CRL Publishing

To test whether you've configured the Certificate Manager correctly to publish certificates and CRLs to the directory, follow these steps:

- Step A. Decide a Directory Entry for Requesting a Certificate
- Step B. Request a Certificate
- Step C. Approve the Request
- Step D. Download the Certificate to the Browser
- Step E. Check if the Directory Has the Certificate
- Step F. Revoke the Certificate
- Step G. Check the Directory for the CRL

Step A. Decide a Directory Entry for Requesting a Certificate

Decide on a user entry for which you will request a certificate. This way, you can check whether the Certificate Manager published the certificate to that entry. The entry you choose could be any end-entity's directory entry, as long as it supports the `userCertificate;binary` attribute.

If you don't have a directory entry yet, you can create one for testing purposes.

Step B. Request a Certificate

The steps outlined below explain how to request a personal certificate from the Certificate Manager using the manual enrollment method (see "Manual Authentication" on page 323). If you've configured the Certificate Manager for automated certificate issuance, for example for directory-based enrollment, you can use the appropriate form and request a certificate.

To request a client or personal certificate from the Certificate Manager:

1. Open a web browser window.
2. Go to the end-entity interface of the Certificate Manager you configured (or to the Registration Manager that's connected to this Certificate Manager). The URL is in this form:

   `https://<host_name>:<end_entity_HTTPS_port>`

   or

   `http://<host_name>:<end_entity_HTTP_port>`
3. In the left frame, under Browser, select Manual.

   This opens the manual enrollment form.
4. Fill in all the values and submit the request.

   The client prompts you to enter the password for your key database.
5. When you enter the correct password, the client generates the key pairs.

   Do not interrupt the key-generation process.

Step C. Approve the Request

Skip this step if you used an automated enrollment method for requesting the certificate. Complete this step if you used the manual enrollment form for requesting the certificate; the request you submitted is waiting in the agent queue for approval by an agent.

To approve the request:

1. Go to the Certificate Manager's Agent Services interface.

   The URL is in this format: `https://<host_name>:<agent_port>`

2. In the left frame, click List Requests.

3. In the form that appears, select the "Show pending requests" option and click Find.

4. In the list of pending requests, locate the request you submitted and approve the request.

   You should see a confirmation page indicating that the certificate has been issued. Don't close the page until after you read the next step.

Step D. Download the Certificate to the Browser

To download the certificate into your browser's certificate database:

1. In the confirmation page, scroll down to the section that says "Installing this certificate in a client."

2. Follow the on-screen instructions and download the certificate to your browser's certificate database.

   (An alternative way to download the certificate is from the Retrieval tab of the end-entity services interface.)

3. Open the browser's security information window and verify that the certificate has been stored in the certificate database.

Step E. Check if the Directory Has the Certificate

If you've configured the Certificate Manager and Directory Server correctly, the Certificate Manager automatically publishes the certificate to the directory whenever it issues a certificate.

Verify that the Certificate Manager has published the certificate to the correct user entry.

If you're using Netscape Directory Server version 4.x you can do this verification from the Directory Server window as follows:

1. In Netscape Console, double-click the Directory Server instance that corresponds to the publishing directory.

   This opens the Directory Server window.

2. Select the Directory tab.

3. Locate the user entry for which you requested the certificate.

4. Double-click the entry and check if the entry has a `certificate` attribute.

   You should find the certificate published to the attribute. You won't be able to see anything interesting about the certificate; it will be a DER-encoded binary blob.

Alternatively, you can point your browser to the user entry in the directory to verify that the certificate has been published. To do this:

1. Open a web browser window.

2. In the URL field, type `ldap://<host_name>:<port>/<base_dn>??sub?(uid=<user_id>)`, substituting `<host_name>` with the fully qualified host name of the Directory Server, `<port_number>` with the port number at which the Directory Server is listening to publishing requests from the Certificate Manager `<base_dn>` with the DN to start searching for the user's entry, and `<user_id>` with the ID of the user to whom you issued the certificate.

   For example, if the directory host name is `corpDirectory`, port number is `389`, base DN is `O=siroe.com`, and user's ID is `jdoe`, the URL would look like this: `ldap://corpDirectory:389/O=siroe.com??sub?(uid=jdoe)`

   In the resulting page, look for the user's certificate-related information. The information typically includes the owner of the certificate, the CA that issued the certificate, the serial number, the validity period, and the certificate fingerprint.

Step F. Revoke the Certificate

To check whether you've configured the Certificate Manager to publish the CRL to the directory correctly, revoke the certificate you issued. In "Step A. Specify CRL Details" on page 824, if you didn't configure the Certificate Manager to

publish the CRL every time a certificate is revoked, go back to the Revocation List tab and select the "Every time a certificate is revoked or taken off-hold" option. After you complete testing, remember to go back to the same tab and uncheck the option.

To revoke the certificate:

1. Go to the end-entity interface for the Certificate Manager (or to the Registration Manager that's connected to this Certificate Manager. Be sure to go to the HTTPS interface (the revocation feature is not available in the HTTP interface).
2. Select the Revocation tab.
3. In the left frame, select User Certificate.

   The User Certificate Revocation form appears.
4. In the Revocation Reason section, select Unspecified and click Submit.

   The client displays the "Select a Certificate" dialog box and prompts you to choose the certificate you want to revoke.
5. Select the certificate you downloaded and click OK.

   The certificate is revoked.

Step G. Check the Directory for the CRL

Verify that the Certificate Manager published the CRL (in this case, containing the single certificate that you revoked) to the correct location in the directory—that is, the `certificateRevocationList;binary` attribute of the CA's entry in the directory.

To do this:

1. Go to the publishing directory.
2. Locate the CA's entry.
3. Check the `certificateRevocationList;binary` attribute.

   You should find the CRL published.

Manually Updating Certificates and CRL in a Directory

Normally you do not need to manually update the directory with certificate-related information; if configured properly, the Certificate Manager handles most of the updates automatically. However, a situation might arise in which you need to update the directory manually. For example, Directory Server might be down for a while and be unable to receive changes from the Certificate Manager. In such a situation, use the forms provided in the Certificate Manager Agent Services interface to manually update the directory.

Certificate Manager's publishing directory can be manually updated by a Certificate Manager agent only. Agent operations are restricted to those with a valid agent certificate; see "Agent's Certificate for SSL Client Authentication" on page 175. For complete details on agent operations, see *Netscape Certificate Management System Agent's Guide*.

Manually Updating Certificates in the Directory

The Update Directory Server form in the Certificate Manager Agent Services interface enables you to manually update the directory with certificate-related information. This form lets you initiate a combination of the following operations:

- Update the directory with certificates.
- Remove expired certificates from the directory.

  Note that you can automate removal of expired certificates from the publishing directory by scheduling an automated job. For details, see "Directory Update and Notification" on page 444.
- Remove revoked certificates from the directory.

To manually update the directory with changes:

1. Open a web browser window.
2. Go to the Certificate Manager Agent Services interface.

   You must submit the proper certificate to get access to this page.

3. Select the Update Directory Server link.

   The Update Directory Server page appears.

4. Select the appropriate options.

5. When you are done specifying the changes that you want updated, click Update Directory.

   The Certificate Manager starts updating the directory with the certificate information in its internal database. In some circumstances, for example if the changes are substantial, updating the directory can take considerable time. During this period, any changes made through the Certificate Manager (for example, any certificates issued or any certificates revoked) may not be included in the update. If you have issued or revoked any certificates during that time, you need to update the directory again to reflect those changes.

   When the directory update is complete, the Certificate Manager displays a status report. If for some reason the process gets interrupted, the server logs an error message. Be sure to check logs if that happens; for details, see "Monitoring Logs" on page 1064.

Manually Updating the CRL in the Directory

The Update Certificate Revocation List form in the Certificate Manager Agent Services interface to enables you to manually update the directory with CRL-related information.

To manually update the CRL information in the directory:

1. Go to the Certificate Manager Agent Services page.

   You must submit the proper client certificate to get access to this page.

2. Select Update Revocation List.

   The Update Certificate Revocation List page appears.

3. From the Signature algorithm drop-down list, select the appropriate signature algorithm.

4. Click Update.

   The Certificate Manager starts updating the directory with the CRL in its internal database. In some circumstances, for example, if the CRL is large, updating the directory may take considerable time. During this period, any changes made to the CRL (for example, any new certificates revoked) may not be included in the update.

   When the directory is updated, the Certificate Manager will display a status report. If the process gets interrupted for some reason, the server logs an error message. Be sure to check logs if that happens; for details, see "Monitoring Logs" on page 1064.

Publishing Certificates and CRLs to Flat Files

The Certificate Manager can publish certificates and CRLs to flat files, which can then be imported into any repository, for example, into a relational database. If you configure the server to publish certificates and CRLs to flat files, it publishes them to files as DER-encoded binary blobs.

- For each certificate the server issues, it creates a file that contains the certificate in its DER-encoded format. Each file is named as `cert-<serial_number>.der`, where `<serial_number>` specifies the serial number of the certificate contained in the file. For example, the filename for a certificate with serial number `1234` will be `cert-1234.der`.

- Every time the server generates the CRL (which could be every time it revokes a certificate and at a regular interval), it creates a file that contains the new CRL in its DER-encoded format. Each file is named as `crl-<this_update>.der`, where `<this_update>` specifies the value derived from the time-dependent variable named `This Update` of the CRL contained in the file. For example, the filename for a CRL with `This Update: Friday January 28 15:36:00 PST 2000`, will be `crl-949102696899.der`.

To configure the Certificate Manager to publish certificates and CRLs to files, follow these steps:

- Step 1. Plan
- Step 2. Configure the Certificate Manager

- Step 3. Test Publishing

Step 1. Plan

Before configuring a Certificate Manager to publish the CA certificate, end-entity certificates, and CRLs to flat files:

- Read sections “Publisher Modules” on page 749 and “Flat File Publisher” on page 752.
- Identify the machine that will contain the DER-encoded files, and create a directory for the files.
- Make sure that the machine has sufficient disk space to accommodate the DER-encoded files that the Certificate Manager will generate; the server generates a file for every certificate it issues and for every CRL it generates. If disk space is a constraint, you can configure the server to create files on two different hosts, one for certificates and another one for CRLs.
- Read “Publishing of CRLs” on page 721. Determine whether you want to publish version 1 or version 2 CRLs. If you decide to publish version 2 CRLs, read “CRL Extension Modules” on page 763 and determine the CRL extensions you want the Certificate Manager to set; you will be required to configure the server to set these extensions.
- Decide the interval for publishing CRLs—configuring the server to publish every time a certificate is revoked will result in that many CRL files.
- Determine the backup media and schedule for these files.

Step 2. Configure the Certificate Manager

To configure a Certificate Manager to publish certificates and CRLs to files, follow these steps:

- Step A. Create a Publisher for the Flat File
- Step B. Create Publishing Rules for Publishing CA Certificate, End-Entity Certificates and CRLs

- Step C. Specify CRL Details
- Step D. Set the CRL Extensions
- Step E. Make Sure Publishing is Enabled

Step A. Create a Publisher for the Flat File

Creating a publisher for the flat file involves creating an instance of the publisher module that enables the Certificate Manager to publish certificates and CRLs to flat files. In the next step, "Step B. Create Publishing Rules for Publishing CA Certificate, End-Entity Certificates and CRLs" on page 844, you specify the publisher you create here.

To create a publisher:

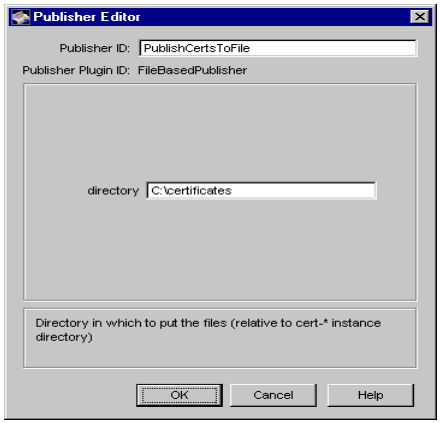
1. Log in to the CMS window for the Certificate Manager (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Certificate Manager, select Publishing, and then select Publishers.

   The right pane displays the Publishers Management tab, which lists configured publisher instances.
4. Click Add.

   The Select Publisher Plugin Implementation window appears. It lists registered publisher modules.
5. Select the module named `FileBasedPublisher`.

   Only this publisher module enables the Certificate Manager to publish certificates and CRLs to flat files; for details about the module, see "Flat File Publisher" on page 752.
6. Click Next.

The Publisher Editor window appears.



7. Enter the appropriate information:

   Publisher ID. Type a name for the rule. Be sure to use an alphanumeric string with no spaces.

   directory. Type the complete path to the directory in which the Certificate Manager should create the DER-encoded files; the path can be an absolute path or can be relative to the CMS instance directory. For example, `C:\certs_crls`.

8. Click OK.

   You are returned to the Publishers Management tab. It should now list the publisher you just created.

9. If you want to publish certificates and CRLs to two separate directories, repeat steps 4 though 8 to create another publisher with the value of the `directory` parameter set to the file path to the other directory.

Step B. Create Publishing Rules for Publishing CA Certificate, End-Entity Certificates and CRLs

Creating a publishing rule for the CRL involves creating a rule that uses the publisher that you created in the previous step.

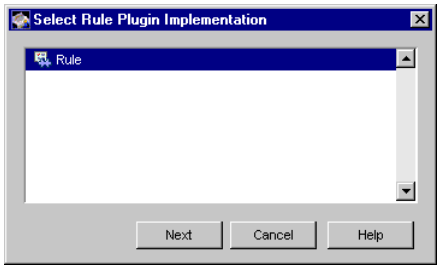
To create a publishing rule:

1. In the navigation tree, under Publishing, select Rules.

   The right pane displays the Rules Management tab, which lists configured publishing rules.

2. Click Add.

   The Select Rule Plugin Implementation window appears. It lists registered modules that enable creating of publishing rules.

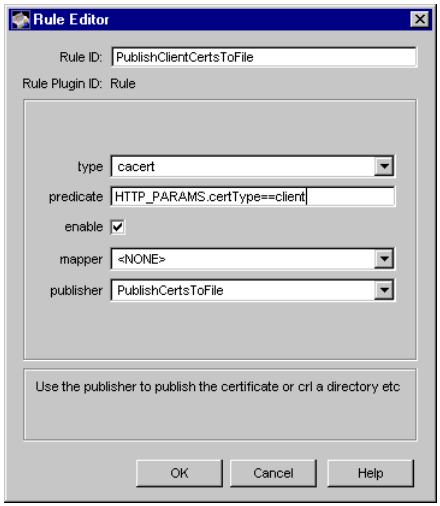


3. Select the module named `Rule`.

   This is the default module. (If you have registered any custom modules, they too will be available for selection.)

4. Click Next.

   The Rule Editor window appears.



5. Enter the appropriate information:

   Rule ID. Type a name for the rule that will help you identify it later; use an alphanumeric string with no spaces.

   type. Select `crl`.

   predicate. Leave this field blank.

   enable. Select this option.

   mapper. Select `<NONE>`.

   publisher. Select the publisher you created in the previous step.

6. Click OK.

   The Rules Management tab appears, listing the new rule you just created for publishing CRLs to flat files.

7. Repeat steps 2 through 6 to create publishing rules for the CA certificate and for each type of end-entity certificates the Certificate Manager will issue. Use Table 22.1 for filling in the correct values in the `type` and `predicate` fields. (For information on predicates, see "Using Predicates in Policy Rules" on page 490.)

Table 22.2 Certificate type and predicate expression

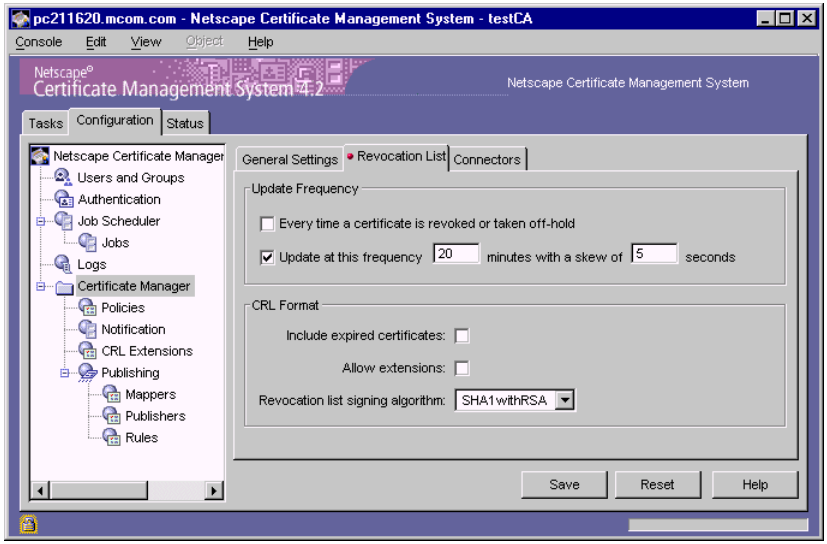
End-entity certificate type	"type" field value	"predicate" field value
SSL client certificate	`certs`	`HTTP_PARAMS.certType==client`
SSL server certificate	`certs`	`HTTP_PARAMS.certType==server`
Object signing certificate	`certs`	`HTTP_PARAMS.certType==objSignClient`
Certificate Manager signing certificate (subordinate CA)	`cacert`	`HTTP_PARAMS.certType==ca`
Registration Manager signing certificate	`certs`	`HTTP_PARAMS.certType==ra`
OCSP responder certificate	`certs`	`HTTP_PARAMS.certType==ocspResponder`
Router certificate	`certs`	`HTTP_PARAMS.certType==CEP-Router`

Step C. Specify CRL Details

You can specify information, such as the publishing interval, the CRL version (whether to include CRL extensions), and the signing algorithm the Certificate Manager should use for signing the CRL object.

To specify CRL details:

1. In the navigation tree, select Certificate Manager, and then in the right pane, select the Revocation List tab.



2. In the Update Frequency section, specify the interval for publishing the CRL to the directory:

   Every time a certificate is revoked, or taken off-hold. Select this option if you want the Certificate Manager to generate the CRL every time it revokes a certificate. Keep in mind that the Certificate Manager attempts to publish the CRL to the configured directory whenever it is generated, in this case, every time a certificate is revoked. Publishing a CRL can be time consuming if the CRL is large. Configuring the Certificate Manager to publish CRLs every time a certificate is revoked may engage the server for a considerable amount of time; during this time, the server will not be able to service any requests it receives and will not be able to update the directory with any changes it receives.

(This setting is not recommended for a standard installation. You can select this option if you want to see the results of revocation immediately, for example, when testing whether the server publishes the CRL to a flat file.)

Update at this frequency. Select this option if you want the Certificate Manager to generate CRLs at regular intervals. In this case, the server publishes the CRL to the configured directory at the interval you specify.

In the adjoining text field, type the interval, in minutes, at which the Certificate Manager should publish CRLs. For example, if you want the server to publish CRLs every day, you should type 1440 in this field.

with a skew of. If you configure the Certificate Manager to update the CRL automatically every time period, the server by default adds a 5 second skew to the next update time to allow time to create the CRL and publish it. For example, if you configure the server to update the CRL every 20 minutes, and if the CRL is updated at 16:00:00, the CRL will be updated again at 16:19:55. You can change the skew by editing the default value, which is specified in seconds.

3. In the CRL Format section, specify the format for publishing the CRL:

   Include expired certificates. Check this box if you want the server to include revoked certificates that have expired in the CRL.

   Allow extensions. Check this box if you want to allow extensions in the CRL. If you enable this option, the server generates and publishes CRLs conforming to X.509 version 2 standard. If you disable this option, the server generates and publishes CRLs conforming to X.509 version 1 standard. By default, the server publishes version 1 CRLs. If you enable this option, be sure to set the required CRL extensions as described in "Step D. Set the CRL Extensions" on page 848.

   Revocation list signing algorithm. Select the algorithm the server should use to sign the CRL. If the Certificate Manager's signing key type is RSA, select `MD2 with RSA`, `MD5 with RSA`, or `SHA-1 with RSA`. If the Certificate Manager's signing key type is DSA, select `SHA-1 with DSA`.

4. To save your changes, click Save.

   The configuration is modified. If the changes you made require you to restart the server, you are prompted accordingly. Don't restart the server yet; you can restart it after you've made all the required changes.

Step D. Set the CRL Extensions

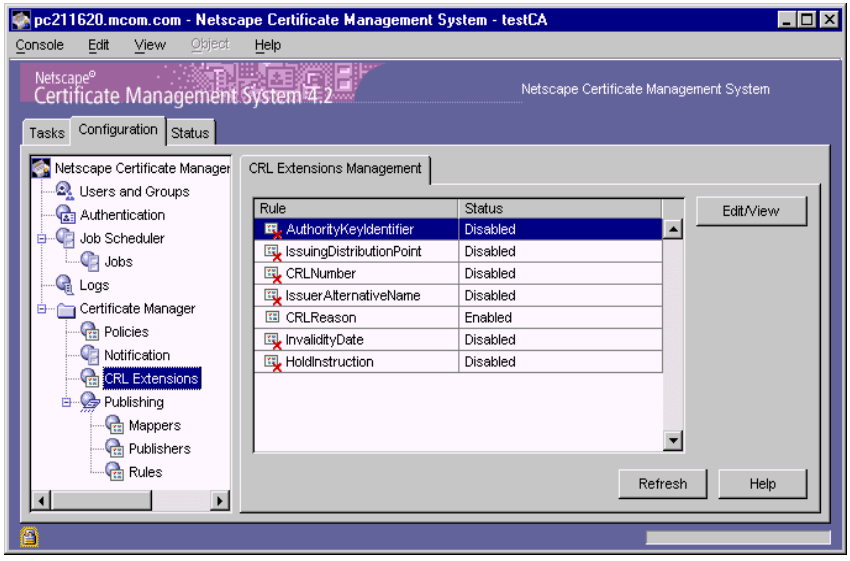
Complete this step only if you configured the Certificate Manager to publish version 2 CRLs in the previous step—that is, if you selected the "Allow extensions" option in "Step C. Specify CRL Details" on page 846.

During installation, the Certificate Manager creates default CRL extension rules; these are shown in Table 21.10 on page 766. Note that the server is configured to add the CRL Reason extension only; all the other rules are in the disabled state. In this step, you modify the default rules to suit your organization's requirements.

To specify the CRL extensions the Certificate Manager should set:

1. In the navigation tree, select Certificate Manager, and then select CRL Extensions.

   The right pane shows the CRL Extensions Management tab, which lists configured extensions.



2. To modify a rule, select it and then click Edit/View.

3. Change the information as appropriate.

   Be sure to supply all the required values. Click the Help button for detailed information on individual parameters.

4. Click OK.

   You are returned to the CRL Extensions Management tab.

5. To modify other rules, repeat steps 2 through 4.

6. Click Refresh to see the updated status of all the rules.

Step E. Make Sure Publishing is Enabled

To make sure that the Certificate Manager is configured for publishing:

1. In the navigation tree, select Certificate Manager, then select Publishing.

   The right pane shows the publishing details necessary for the server to publish to an LDAP-compliant directory, to flat files, or to an online validation authority.

2. Make sure that the Enable Publishing option is selected. If it is already selected, leave it as it is. If it isn't, select it.

   (Leave the "Enable default LDAP connection" option as it is; it specifies that the Certificate Manager is configured to publish certificates and CRLs to an LDAP directory.)

3. If you changed anything, click Save to save the changes.

   If the changes you made require you to restart the server, you are prompted accordingly. In that case, restart the server.

Step 3. Test Publishing

To verify that the Certificate Manager is publishing certificates and CRLs correctly to flat files, follow these steps:

- Step A. Request a Certificate
- Step B. Approve the Request
- Step C. Download the Certificate to the Browser
- Step D. Check the File for the Certificate
- Step E. Revoke the Certificate

- Step F. Check the File for the CRL

Step A. Request a Certificate

The steps outlined below explain how to request a personal certificate from the Certificate Manager using the manual enrollment method (see "Manual Authentication" on page 323). If you've configured the Certificate Manager for automated certificate issuance, for example for directory-based enrollment, you can use the appropriate form and request a certificate.

To request a client or personal certificate from the Certificate Manager:

1. Open a web browser window.
2. Go to the end-entity interface of the Certificate Manager you configured (or to the Registration Manager that's connected to this Certificate Manager). The URL is in this form:

   `https://<host_name>:<end_entity_HTTPS_port>`

   or

   `http://<host_name>:<end_entity_HTTP_port>`
3. In the left frame, under Browser, click Manual.

   This opens the manual enrollment form.
4. Fill in all the values and submit the request.

   The client prompts you to enter the password for your key database.
5. When you enter the correct password, the client generates the key pair.

   Do not interrupt the key-generation process.

Step B. Approve the Request

Skip this step if you requested the certificate using any of the automated enrollment methods in "Step A. Request a Certificate" on page 851. Complete this step if you used the manual enrollment form to request the certificate; the request you submitted is waiting in the agent queue for approval by an agent.

To approve the request:

1. Go to the Certificate Manager's Agent Services interface.

   The URL is in this format: `https://<host_name>:<agent_port>`

2. In the left frame, click List Requests.

3. In the form that appears, select the "Show pending requests" option and click Find.

4. In the list of pending requests, identify the request you submitted and approve the request.

   You should see a confirmation page indicating that the certificate has been issued. Don't close the page until after you complete the next step.

Step C. Download the Certificate to the Browser

To download the certificate into your browser's certificate database:

1. In the confirmation page, scroll down to the section that says "Installing this certificate in a client."

2. Follow the on-screen instructions and download the certificate to your browser's certificate database.

   (An alternative way to download the certificate is from the Retrieval tab of the end-entity services interface.)

3. Open the browser's security information window and verify that the certificate has been stored in the certificate database.

Step D. Check the File for the Certificate

Whenever the Certificate Manager issues a certificate, it automatically attempts to publish the certificate to the configured repository—in this case, the flat file. To check whether the Certificate Manager published the correct certificate, you need to do the following:

1. Check whether the server generated the DER-encoded file containing the certificate.

   To check whether the server published the certificate as a binary blob to the specified directory, go to the directory or folder you specified for the server to publish certificates. You should see a file with name similar to `cert-<serial_number>.der`, where `<serial_number>` specifies the serial number of the certificate contained in the file. If you don't see a file, check your configuration.

2. Convert the DER-encoded certificate to its base 64-encoded format using the Binary to ASCII tool (see "Binary to ASCII Tool" on page 1189).

   To convert the DER-encoded certificate to its base 64-encoded form:

   1. Open a command window.

   2. Go to this directory: `<server_root>/bin/cert/tools`

   3. At the prompt, enter this: `BtoA[.bat] <input_file> <output_file>`

      substituting `<input_file>` with the path to the file that contains the DER encoded certificate and `<output_file>` with the path to the file to write the base-64 encoded certificate. (The optional `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.)

      For example, if the file is in `C:\certificates\cert-1234.der` and you want the base-64 encoded certificate to be in `C:\certificates\cert-1234.txt`, the command would look like this: `BtoA C:\certificates\cert-1234.der C:\certificates\cert-1234.txt`

   4. When the conversion is complete, open the `cert.txt` file in a text editor. You should see a base-64 encoded certificate similar to this:

```
-----BEGIN CERTIFICATE-----
MMIIBtgYJYIZIAYb4QgIFoIIBpzCCAZ8wggGbMIIBRaADAgEAAgEBMA0GCSqGSIb3DQEBBAUAM
FcxCAJBgNVBAYTAlVTMSwwKgYDVQQKEyNOZXRzY2FwZSBDb21tdW5pY2F0aWhfyyuougjgjjgm
kgjkgmjgfjfgjjjgfyjfyj9ucyBDb3Jwb3JhdGlvbjpMEaMBgGA1UECxMRSXNzdWluZyhgdf
```

```
    hbfdpffjphotokogdhkBBdXRob3JpdHkwHhcNOTYxMTA4MDkwNzM0WhcNOTgxMTA4MDkwNzMM0
    WjBXMQswCQYDVQQGEwJVUzEsMCoGA1UEChMjTmV0c2NhcGUgQ29tbXVuaWNhdGlvbnMgQ29ycG
    9yY2F0aW9ucyBDb3Jwb3JhdGlvbjpMEaMBgGA1UECxMRSXNzdWluZyBBdXRob3JpdHkwHh
-----END CERTIFICATE-----
```

3. Convert the base 64-encoded certificate to a human-readable form using the Pretty Print Certificate tool (see "Pretty Print Certificate Tool" on page 1190).

   To convert the base 64-encoded certificate to a human-readable form:

   1. Check the command window to make sure that you are in this directory: `<server_root>/bin/cert/tools`

   2. At the prompt, enter this:

      ```
      PrettyPrintCert[.bat] <input_file> [<output_file>]
      ```

      substituting `<input_file>` with the path to the ASCII file that contains the base-64 encoded certificate and `<output_file>` with the path to the file to write the certificate in a human-readable form. If you don't specify an output file, the certificate information is written to the standard output. (The optional `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.)

      For example, if the base-64 encoded certificate is in `C:\certificates\cert-1234.txt` and you want the human-readable form of the certificate to be displayed on your screen, the command would look like this:

      ```
      PrettyPrintCert.bat C:\certificates\cert-1234.txt
      ```

      When the conversion is complete, you should see the certificate you issued in human-readable form.

   3. Compare the output with the certificate you issued; be sure to check the serial number in the certificate with the one used in the filename.

      If everything matches, the Certificate Manager is configured correctly to publish certificates to files.

Step E. Revoke the Certificate

To check whether the Certificate Manager is configured correctly to publish CRLs to flat files, you need to revoke the certificate you issued. Before revoking the certificate, make sure that you've configured the Certificate Manager to publish the CRL every time a certificate is revoked. (In "Step C. Specify CRL Details" on page 846, if you didn't configure the Certificate Manager to publish

the CRL every time a certificate is revoked, go back to the Revocation List tab and check the "Every time a certificate is revoked or taken off-hold" option. After the testing, remember to go back to the same tab and uncheck the option.)

To revoke the certificate:

1. Go back to the end-entity interface for the Certificate Manager (or to a Registration Manager that's connected to this Certificate Manager. Be sure to go to the HTTPS interface; the revocation feature is not available in the HTTP interface.
2. Click the Revocation tab.
3. In the left frame, click User Certificate.

   The User Certificate Revocation form appears.
4. In the Revocation Reason section, select Unspecified and click Submit.

   The browser displays the "Select a Certificate" dialog box and prompts you to choose the certificate you want to revoke.
5. Select the certificate you downloaded and click OK.

   The certificate is revoked.

Step F. Check the File for the CRL

Whenever the Certificate Manager generates a CRL, it automatically attempts to publish the CRL to the configured repository—in this case, the flat file. The CRL it publishes is a binary blob, in the DER-encoded format. To check whether the Certificate Manager published the correct CRL (in this case, the CRL contains only one certificate), you need to do the following:

1. Check whether the server generated the DER-encoded file containing the CRL.

   To check whether the server published the CRL as a binary blob to the specified directory, go to the directory you specified for the server to publish CRLs. You should find a file with its name in the `crl-<this_update>.der` format, where `<this_update>` specifies the value derived from the time-dependent variable named `This Update` of the CRL contained in the file. If you don't see the file, check your configuration.

2. Convert the DER-encoded CRL to its base 64-encoded format using the Binary to ASCII tool (see "Binary to ASCII Tool" on page 1189).

   To convert the DER-encoded CRL to its base 64-encoded form:

   1. Open a command window.
   2. Go to this directory: `<server_root>/bin/cert/tools`
   3. At the prompt, enter this: `BtoA[.bat] <input_file> <output_file>`

      substituting `<input_file>` with the path to the file that contains the DER-encoded CRL and `<output_file>` with the path to the file to write the base-64 encoded CRL. (The optional `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.)

      For example, if the DER-encoded file is in `C:\crls\crl-949102696899.der` and you want the base-64 encoded CRL to be in `C:\crls\crl-949102696899.txt`, the command would look like this:

      ```
      BtoA C:\crls\crl-949102696899.der C:\crls\crl-
      949102696899.txt
      ```

   4. When the conversion is complete, open the `crl.txt` file in a text editor. You should see a base-64 encoded CRL similar to this:

      ```
      -----BEGIN CRL-----

      MIIBkjCBAIBATANBgkqhkiG9w0BAQQFADAsMREwDwYDVQQKEwhOZXRzY2FwZTEXMBUGA1UEAxM
      OQ2VydDQwIFRlc3QgQ0EXDTk4MTIxNzIyMzcyNFowgaowIAIBExcNOTgxMjE1MTMxODMyWjAMM
      AoGA1UdFQQDCgEBMCACARIXDTk4MTIxNTEzMjA0MlowDDAKBgNVHRUEAwoBAjAgAgERFw05ODE
      yMTYxMjUxNTRaMAwwCgYDVR0VBAMKAQEwIAIBEBcNOTgxMjE3MTAzNzI0WjAMMAoGA1UdFQQDC
      gEDMCACAQoXDTk4MTEyNTEzMTExOFowDDAKBgNVHRUEAwoBATANBgkqhkiG9w0BAQQFAAOBgQB
      CN85O0GPTnHfImYPROvoorx7HyFz2ZsuKsVblTcemsX0NL7DtOa+MyY0pPrkXgm157JrkxEJ7G
      BOeogbAS6iFbmeSqPHj8+JBH5stJNnfTCuhaM6Wx63Wc9LwZXOXTPsvpGxq0YYI0+DPfBZlI3z
      4lCsNczxJV+9NkeMrheEg==

      -----END CRL-----
      ```

3. Convert the base 64-encoded CRL to a human-readable form using the Pretty Print CRL tool (see "Pretty Print CRL Tool" on page 1193).

To convert the base 64-encoded CRL to a human-readable form:

1. Check the command window to make sure that your are at this directory: `<server_root>/bin/cert/tools`

2. At the prompt, enter this: `PrettyPrintCrl[.bat] <input_file> [<output_file>]`

   substituting `<input_file>` with the path to the ASCII file that contains the CRL in its base 64-encoded format and `<output_file>` with the path to the file to write the CRL information in a human-readable form. If you don't specify an output file, the CRL information is written to the standard output. (The optional `.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.)

   For example, if the base-64 encoded CRL is in `C:\crls\crl-949102696899.txt` and you want the human-readable form of the CRL to be displayed on your screen, the command would look like this:

   ```
   PrettyPrintCrl.bat C:\crls\crl-949102696899.txt
   ```

   When the conversion is complete, you should see the CRL (in this case, the CRL will only contain the certificate you revoked) in the human-readable form.

3. Compare the output with the certificate you revoked.

   If they match, the Certificate Manager is configured correctly to publish CRLs to files.

Publishing CRLs to Online Validation Authority

You can configure the Certificate Manager to publish CRLs to an online certificate validation authority, such as ValiCert Certificate VA™ (Certificate VA). This section explains how to set up Certificate VA (included with Certificate Management System) as your local online validation authority.

- Step 1. Plan
- Step 2. Install an OCSP-Compliant Client

- Step 3. Install the Certificate VA
- Step 4. Configure Certificate Manager for Required Extension Policies
- Step 5. Replace the Certificate VA's Certificate
- Step 6. Restart Certificate VA
- Step 7. Configure the Certificate Manager for Publishing CRLs
- Step 8. Test Publishing

Step 1. Plan

Before you configure a Certificate Manager to publish CRLs to Certificate VA, do the following:

- If you are unfamiliar with Online Certificate Status Protocol (OCSP), read the PKIX draft RFC 2560 available at this site:
  `http://www.ietf.org/rfc/rfc2560.txt`
- Read these sections: "Publishing of CRLs to an Online Validation Authority" on page 726 and "ValiCert Publisher" on page 759.
- Check whether you have deployed any OCSP-compliant clients. If you haven't, you can use the OCSP-compliant security plug-in module for Netscape Communicator, Netscape Personal Security Manager.
- Identify a host machine for installing Certificate VA.
- Check whether you've enabled Certificate Manager's end-entity HTTP port; see "Configuring Port Numbers" on page 158. If the end-entity HTTP port is enabled, note whether the port number assigned is 80 (which is the default). If it is, make sure you assign a different pot number to Certificate VA during its installation in "Step 3. Install the Certificate VA" on page 861; (The default service port for Certificate VA is also 80.)

Step 2. Install an OCSP-Compliant Client

If you decided to install Personal Security Manager, which came with Certificate Management System, follow the steps below. If you downloaded the latest version of Personal Security Manager from the web site, follow the instructions that came with it. If you don't want to install Personal Security Manager, skip to "Step 3. Install the Certificate VA" on page 861.

1. Go to this directory: `<server_root>/psm11`

   Make sure you see the following files:

   - `cmcjavascriptapi.html`—describes a JavaScript API for performing user certificate management operations within a client. The JavaScript runs in the context of an enrollment page served by a Certificate Manager or Registration Manager, enabling it to instruct the client to perform PKI operations, such as key generation, certificate-request generation, key archival, import of user certificates, key recovery, and revocation requests.

   - `release_notes.html`—explains how to install the product and lists known issues and restrictions. You must read this first for installation instructions.

   - `psm11_linux2.1.tar.gz`— use this for installing Personal Security Manager on a Linux machine.

   - `psm11_solaris2.5.1.tar.gz`—use this for installing Personal Security Manager on a Unix machine (Sun Solaris).

   - `psm11_win32.jar`—use this for installing Personal Security Manager on a Windows NT machine.

2. Copy the file appropriate to the machine on which you have Netscape Communicator, version 4.7 or later, installed.

3. Follow the instructions in the release notes and install the product.

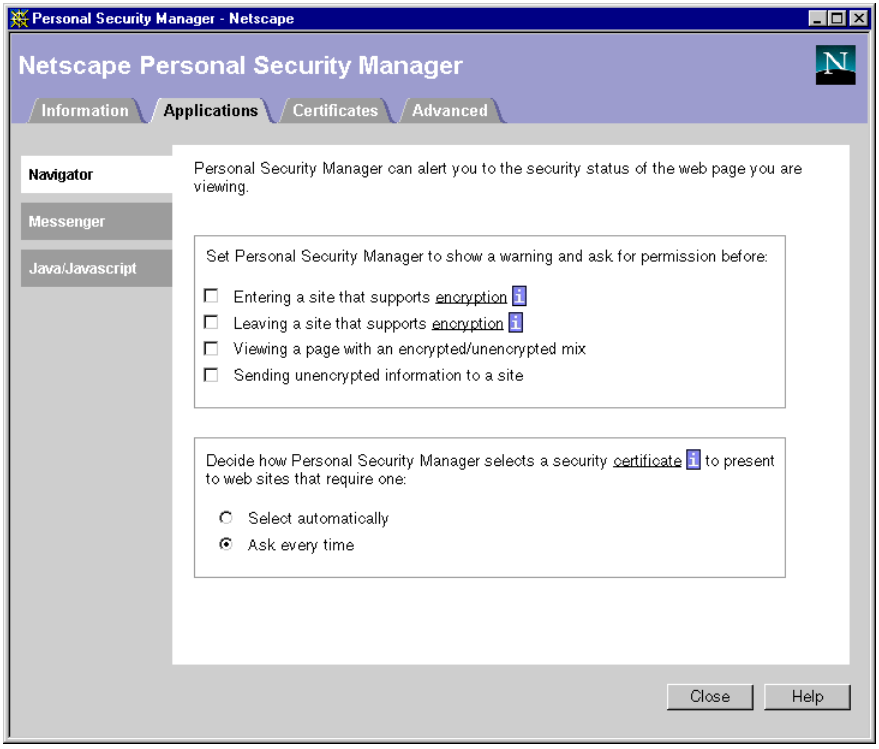
   For example, in a Windows NT system, you can install Personal Security Manager by entering the path to the `psm11_win32.jar` file in the browser's URL area. On a Solaris system, you can unzip the file by running `gunzip`

`psm_11_solaris2.5.1.tar.gz`, untar the file by running `tar xvf psm11_solaris2.5.1.tar`, and then install Personal Security Manager by running `psm-install`.

4. Verify that Personal Security Manager is installed.

   In the menu bar, click Communicator, and from the Tools menu, select Security Info. You should see the Personal Security Manager interface (Figure 22.1).

Figure 22.1 Personal Security Manager interface within Netscape Communicator



Step 3. Install the Certificate VA

To install the Certificate VA, follow these steps:

- Step A: Verify and Copy Files

- Step B. Read the Documentation
- Step C. Run the Installation Program
- Step D. Generate a Key Pair and Self-Signed Certificate
- Step E. Copy the CA Certificate
- Step F. Add the CA Certificate to the Certificate Store

Step A: Verify and Copy Files

To do this:

1. Go to this directory: `<server_root>/cva301`
2. Make sure you see these files:
   - `docs` directory—contains the product documentation.
   - `release_notes.txt`—lists known issues and restrictions.
   - `cva301readme.txt`—contains information about the Certificate VA. You must read this first for instructions.
   - `CVA30setup.exe`—executable for installing the Certificate VA on a Windows NT machine; the executable is available only in the Windows NT package of Certificate Management System.
   - `cva30setup.tar`—executable for installing the Certificate VA on a Sun Solaris machine; the executable is available only in the Unix package of Certificate Management System.
3. Copy files to the machine on which you want to install your OCSP responder.

Step B. Read the Documentation

Read the documentation: release notes for the package (`release notes.txt`), readme for Certificate VA, version 3.01 (`cva301readme.txt`), and the *ValiCert Certificate VA Installation and Administration Guide*, version 3.01 (`cva.pdf` file available in the `docs` directory).

Step C. Run the Installation Program

To install Certificate VA, follow the instructions in Chapter 2 of *ValiCert Certificate VA Installation and Administration Guide*. Be sure to do the following:

1. When you run the setup (or install) program, you're prompted to specify the following:

   - Host name—the Certificate VA's host name; by default, it is the name of the machine in which you install Certificate VA.
   - Service port—the port number at which Certificate VA listens to certificate-validation requests from OCSP-compliant clients; by default, it is 80.
   - Administration port—the port number at which Certificate VA listens to administration requests; by default, it is 13333.

   Note the values you assign to these.

2. When the setup (or install) program is complete, choose the option to launch the Administration Interface.

   The installation program opens the Certificate VA Management menu (which is at `http://<host>:<administration_port>/index.html`) in a browser window.

Step D. Generate a Key Pair and Self-Signed Certificate

The section briefly outlines the steps you need to follow for Certificate VA to work with Certificate Management System; general details about this procedure can be found in Chapter 3 of *ValiCert Certificate VA Installation and Administration Guide*.

To generate a key pair and self-signed certificate:

1. In the left pane, under SETUP, click Create New Key Pair, specify the details, and generate a key pair.

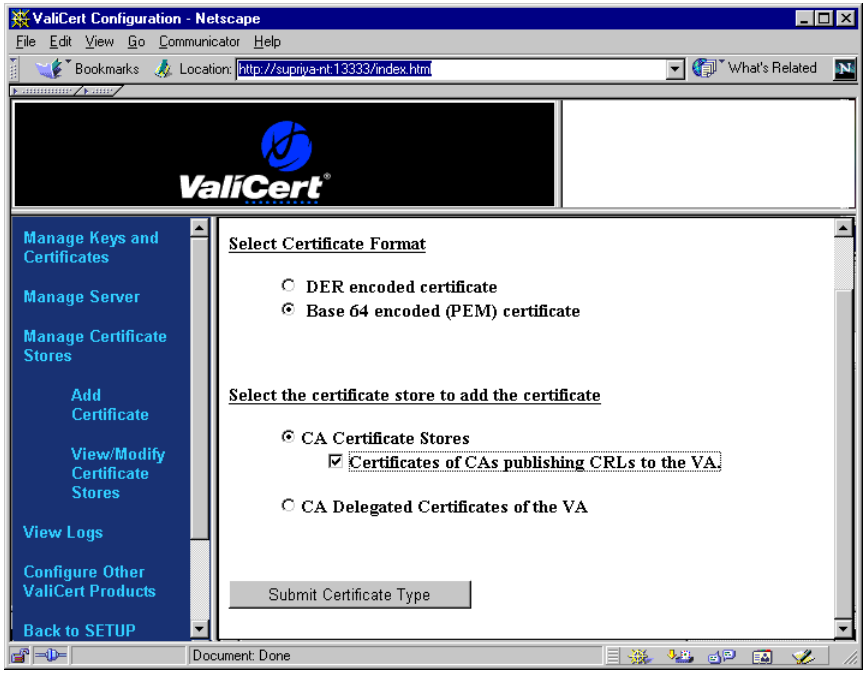
2. Next, specify values for the certificate subject name, select the "Generate self-signed certificate" option, and submit the request. (You must select this option.)

3. Scroll down the page and click Next Step.

4. In the Add Certificate form that appears, select the following:

   - In the Select Certificate Format section, select the "Base 64 encoded (PEM) certificate" option.

   - In the "Select the certificate store to add the certificate" section, select the "CA Certificates Stores" option and then select the "Certificates of CAs publishing CRLs to the VA" option.



5. Click Submit Certificate Type.

The form for pasting the CA certificate (in its base 64-encoded format) shows up. Don't close the form as you will need to use it in "Step F. Add the CA Certificate to the Certificate Store" on page 864.

Step E. Copy the CA Certificate

In order for the OCSP-compliant clients in your PKI to successfully validate the OCSP responder certificate presented by Certificate VA, the Certificate Manager's CA signing certificate must be installed in the Certificate VA's certificate store. The CA signing certificate is available in the Certificate Manager's internal database.

To copy the CA signing certificate:

1. Open another browser window.
2. Go to the Certificate Manager's end-entity interface.
3. Select the Retrieval tab.
4. In the left frame, click List Certificates.
5. Click the Find button.
6. Click the Details button for the first certificate listed; this is the CA's signing certificate.
7. Copy the base-64 encoded certificate, including the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` marker lines, to the clipboard or to a text file.

Step F. Add the CA Certificate to the Certificate Store

To install the CA certificate you copied in the Certificate VA's certificate store:

1. Go back to the browser window that is showing the Certificate VA interface for pasting the CA signing certificate.
2. Paste the certificate to the text area. Be sure to include the marker lines, `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`.

3. Click Submit Certificate.

   A page appears indicating whether the certificate has been added successfully.

4. Click Next Step.

5. Scroll to the bottom of the page.

6. Click Submit Configuration Parameters.

7. Click Next Step.

8. Reload the Certificate VA Management menu page (hold the Shift key down and click the browser's Reload icon on the menu bar).

   Don't close the page.

Step 4. Configure Certificate Manager for Required Extension Policies

As a part of setting up an OCSP-compliant PKI setup, you will be requesting a OCSP responder certificate for Certificate VA (you installed in the previous step) from the Certificate Manager. For this certificate to work properly, it must contain the following extensions:

- `OCSPNoCheck`—Presence of this extension indicates that an OCSP client should not use OCSP to check the revocation status of the OCSP responder certificate, because the certificate is only used to identify the responder that does the checking. (This extension is required to avoid a circular reference.) For details about this extension, see "OCSP No Check Extension Policy" on page 653.

- `OCSPSigning`—This is an *Extended Key Usage* extension with a unique value, `OCSPSigning`. Presence of this extension indicates that the key pair that corresponds to the certificate used by the OCSP responder can be used for signing OCSP responses. For details about this extension, see "OCSPSigningExt Rule" on page 603.

Also, for testing whether your OCSP-compliant clients can verify revocation status of certificates by querying the OCSP responder, you will be issuing a client certificate containing the *Authority Information Access* extension to

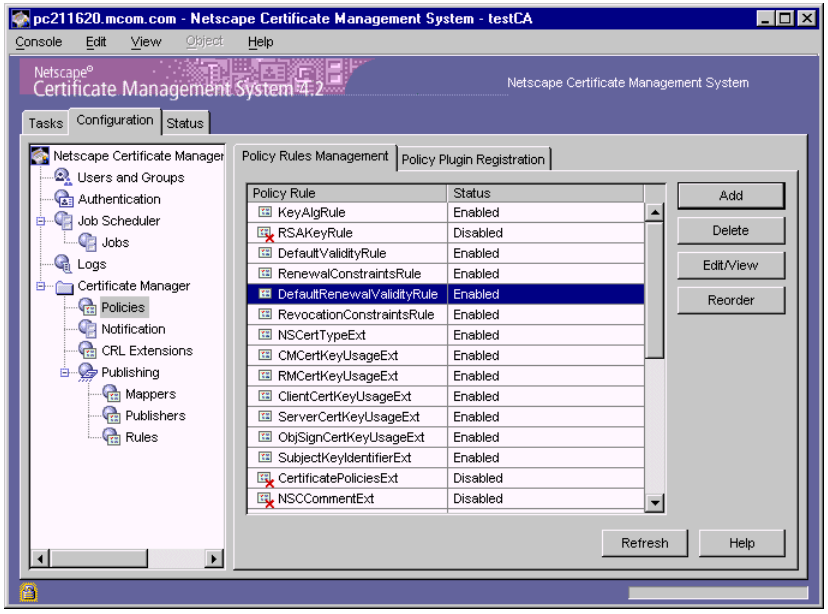
Personal Security Manager you installed. The extension specifies the location of the online validation authority, in this case, Certificate VA. For details about this extension, see “Authority Information Access Extension Policy” on page 558.

The Certificate Manager can add an extension to a certificate it issues only if the corresponding policy is enabled and configured properly. Hence, before issuing the OCSP responder and OCSP-compliant client certificate, you must verify that the Certificate Manager is configured with the appropriate policy rules to add the required extensions to these certificates (which you will request in the steps that follow).

To verify the status of policy rules that enable the Certificate Manager to add the extensions required in the OCSP responder certificate and the OCSP-compliant client certificate:

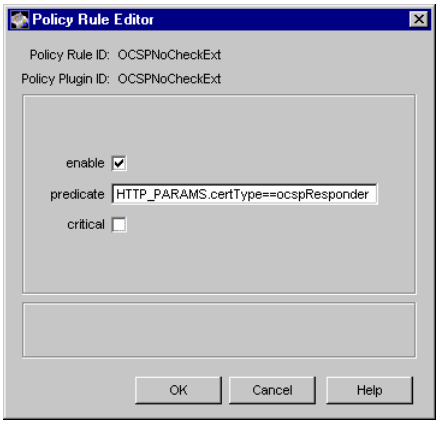
1. Log in to Netscape Console (see “Logging In to Netscape Console” on page 69).
2. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).
3. In the navigation tree, select Certificate Manager, and then select Policies.

   The Policy Rules Management tab appears. It lists configured policy rules.



4. In the Policy Rule list, select the rule named `OCSPNoCheckExt` and click Edit; this rule was created by default during installation.

   The Policy Rule Editor window appears, showing how this rule is currently configured.



5. Make sure the values assigned to parameters are as follows:

   Enable. Checked.

   predicate. `HTTP_PARAMS.certType==ocspResponder`.

   critical. Unchecked.

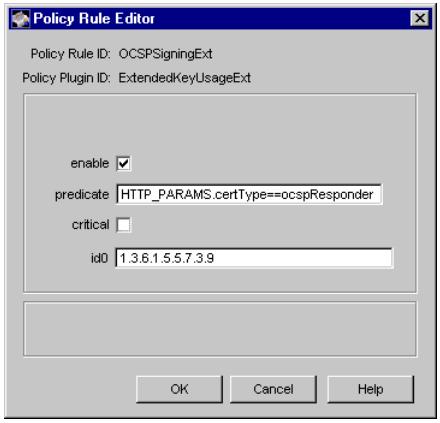
   If you need details about any of these parameters, click the Help button or see Table 18.21 on page 656.

6. Click OK.

   You are returned to the Policy Rules Management tab.

7. Select the rule named `OCSPSigningExt` and click Edit; this rule was created by default during installation.

The Policy Rule Editor window appears, showing how this rule is configured.



8. Make sure the values assigned to parameters are as follows:

   Enable. Checked.

   predicate. `HTTP_PARAMS.certType==ocspResponder`.

   critical. Unchecked.

   id0. `1.3.6.1.5.5.7.3.9`.

   If you need details about any of these parameters, click the Help button or see Table 18.10 on page 600.

9. Click OK.

   You are returned to the Policy Rules Management tab.

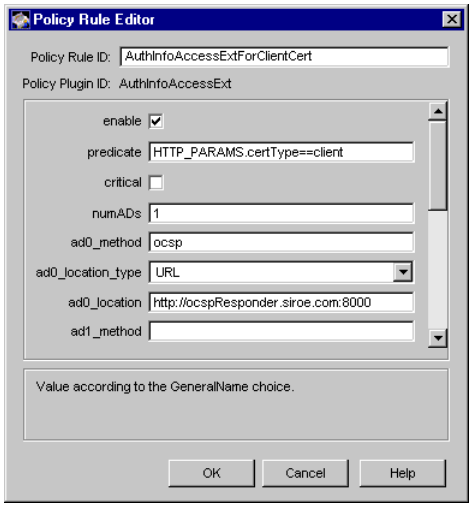
10. Click Add.

    The Select Policy Plugin Implementation window appears. It lists registered policy modules.

11. Select the module named `AuthInfoAccessExt` and click Next.

    The Policy Rule Editor window appears. It lists the configuration information required for this policy rule.

12. Assign the following values:



Enable. Check this box.

predicate. Type `HTTP_PARAMS.certType==client`.

critical. Leave this option unchecked.

numADs. Type `1`.

ad0_method. Type `ocsp` or `1.3.6.1.5.5.7.48.1`.

ad0_location_type. Select `URL`.

ad0_location. Type the complete path to the location where your OCSP responder listens to calls from OCSP-compliant clients. The path should be in this format: `http://<host_name>:<service_port>`.

For example, if the URL to the Certificate VA you installed is `http://ocspResponder.siroe.com:8000`, type that URL here.

If you need details about any of these parameters, click the Help button or see Table 18.2 on page 562.

13. Click OK.

    You are returned to the Policy Rules Management tab.

14. Click Refresh.

The Certificate Manager is ready to request OCSP responder and client certificates.

Step 5. Replace the Certificate VA's Certificate

When you installed Certificate VA (in "Step 3. Install the Certificate VA" on page 861), you generated a key pair and a corresponding certificate; Certificate VA self signed this certificate. The certificate is intended for use by Certificate VA to identify itself to OCSP-compliant clients and the private key of the key pair is intended for use by Certificate VA to sign certificate-validation responses it will send when queried about the revocation status of a certificate.

For Certificate VA to function as your CA-designated responder, you must replace its self-signed certificate with an OCSP responder certificate issued by your CA—the Certificate Manager that signs or revokes the certificates whose revocation status will be verified by the OCSP responder. You added the CA signing certificate of this CA to the Certificate VA's certificate store.

The end-entity interface of both Registration Manager and Certificate Manager include a form that allows you to manually request a certificate for the OCSP responder. The default enrollment form provided for requesting an OCSP Responder certificate includes all the attributes (for example, `HTTP_PARAMS.certType=ocspResponder`) that identify it as a OCSP responder certificate. When the certificate request passes through Certificate Manager's policies, the required extensions (`OCSP no check` and `OCSP signing`) get added to the certificate.

To convert the Certificate VA to your CA-designated responder:

- Step A. Copy the Server's Certificate Signing Request
- Step B. Request an OCSP Responder Certificate From the Certificate Manager
- Step C. Approve the Request
- Step D. Add the Certificate to the Certificate Store
- Step E. Verify That the Certificates Are Stored

Step A. Copy the Server's Certificate Signing Request

To copy the Certificate VA's certificate signing request (CSR):

1. Go to the Certificate VA Management menu.

   In the browser's URL field, enter the URL for the Certificate VA Management menu; it's in this form: `http://<host_name>:<administration_port>`

2. In the left frame, select Manage Keys and Certificates, and then select the Display Certificate Request option.

3. Select the type of request and click Display Certificate Request.

   The certificate signing request, in its base 64-encoded format, appears.

4. Copy the base 64-encoded blob, including the marker lines, `-----BEGIN CERTIFICATE REQUEST-----` and `-----END CERTIFICATE REQUEST-----`, to the clipboard or a text file. For example, the information you copied should look like this:

   ```
   -----BEGIN CERTIFICATE REQUEST-----
   XRpb24xGjAYBgNVBAsTEUlzc3VpbmcgQXV0aG9yaXR5MFowDQYJKoZIhvcNAQEBBQADSQAwRgJBAM
   OBiQPcK8851jjQXA2GBsaKNFg6pYaM3qhQhM0w5EIy6P1ttMjc5MlPIzZHdlgNdQLzaNoLMVKjOV5
   sBp+ffkCAQMwDQYJKoZIhvcNAQEEBQADQQCWPU4gI5uaWM3Egs9909HRGIHGgwpR7Y538BGDTHOGD
   KBGDKBNGDKHJPYRKJOKNXKCQWUY7P0Y9=E50668695SWRGQ3NRI3QJR3QPIGIGIRWGAbXfhQMMIIB
   BzCBsgIBADBPMQswCQYDVQQGEwJVUzEoMCYGA1UEChMfTmV0c2NhcGUgRGlyZWN0b3J5IFB1YxpY2
   F0aW9uczEWMBQGA1UEAxMNZHVtcC5tY29tLmNvbTBaMA0GCSqGSIb3DQEBAQU2nfjiMEYCQQ0ksMR
   aLGdfp4m0OiGcgijG5KgOsyRNvwGYW7kfW+8mmijDtZRjYNjjcgpF3VnlsbxbclX9LVjjNLCM57u3
   7XZdAgEDoAAwDQYJKoZIhvcNAQEEBQADQQCYUTnUtCVGyNrYGSfydclqiovxy1fRD1z23zg+eK7n8
   5UyE4r5zGZjDsMYr172ytfAFL7DeG83DWzr8Z
   -----END CERTIFICATE REQUEST-----
   ```

   Don't make any changes to the copied information. You need to paste the request exactly as it is into the OCSP Responder Enrollment form of the Certificate Manager.

Step B. Request an OCSP Responder Certificate From the Certificate Manager

To request an OCSP responder certificate from the Certificate Manager:

1. Go to the end-entity interface of the Certificate Manager that will publish the CRL to Certificate VA; if you've disabled enrollment via the Certificate Manager, go to the Registration Manager that's connected to this Certificate Manager. The URL is in one of these forms:

   `https://<host_name>:<end_entity_HTTPS_port>`

   or

   `http://<host_name>:<end_entity_HTTP_port>`

2. In the left frame, under Server, click OCSP Responder Enrollment.

   This opens the form for requesting a certificate for an OCSP responder.

3. Complete the form with the information that the Certificate Manager needs to create a certificate for Certificate VA.

   PKCS #10 Request. In the certificate request text area, paste the certificate signing request, including the `-----BEGIN CERTIFICATE REQUEST-----` and `-----END CERTIFICATE REQUEST-----` marker lines, that you copied to the text file.

   Contact Information. In this section, type your name, email address, and phone number. These values will be used by the CA, if the need arises. Be sure to enter your email address. This is the address where the CA will send the certificate once it has been issued (if the automated-notification feature explained in "Notifications of Certificate Issuance to End Entities" on page 450 is turned on).

   Additional Comments. Type any information that will help the issuing agent who will process the request (if you aren't that agent). For example, you might want to enter the name of the person who instructed you to obtain a certificate or some other administrative information.

4. Click Submit.

   The Certificate Manager confirms that your request was submitted.

Step C. Approve the Request

The enrollment method for an OCSP responder certificate is manual, which means that the request gets queued for agent approval.

To approve the request you submitted (you'll need agent privileges to do this):

1. Go to the Certificate Manager Agent Services interface.

   The URL is in this format: `https://<host_name>:<agent_port>`

2. In the left frame, click List Requests.

3. In the form that appears, select the "Show pending requests" option and click Find.

4. In the list of pending requests, locate the request and click Details.

5. Check the information and click Do It.

   If your request contains all the information required by the Certificate Manager to issue a certificate, the server responds that it has issued the certificate, shows the Certificate VA's certificate, and tells how to install the new certificate.

   Scroll down to the section where the certificate is shown in its base 64-encoded format; it will look similar to the sample below:

   ```
   -----BEGIN CERTIFICATE-----
   MMIIBtgYJYIZIAYb4QgIFoIIBpzCCAZ8wggGbMIIBRaADAgEAAgEBMA0GCSqGSIb3DQEBBAUAMFcx
   CAJBgNVBAYTAlVTMSwwKgYDVQQKEyNOZXRzY2FwZSBDb21tdW5pY2F0aWhfyyuougjgjjgmkgjkgm
   jgfjfgjjjgfyjfyj9ucyBDb3Jwb3JhdGlvbjpMEaMBgGA1UECxMRSXNzdWluZyhgdfhbfdpffjpho
   tokogdhkBBdXRob3JpdHkwHhcNOTYxMTA4MDkwNzM0WhcNOTgxMTA4MDkwNzMM0WjBXMQswCQYDVQ
   QGEwJVUzEsMCoGA1UEChMjTmV0c2NhcGUgQ29tbXVuaWNhdGlvbnMgQ29ycG9yY2F0aW9ucyBDb3J
   wb3JhdGlvbjpMEaMBgGA1UECxMRSXNzdWluZyBBdXRob3JpdHkwHh
   -----END CERTIFICATE-----
   ```

6. Copy the certificate, including the marker lines `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`, to the clipboard or to a text file. In the next step, you'll be required to paste this into another form.

If you lose the certificate blob inadvertently, here's how you can get it again:

1. In the Agent Services interface, click List Certificates or Search for Certificates.

2. Search for the certificate you just issued.

Step D. Add the Certificate to the Certificate Store

After you get an OCSP responder certificate, you should install it in Certificate VA's certificate store. This way, Certificate VA can use the certificate to identify itself as a CA -designated OCSP responder.

To install the OCSP responder certificate in the Certificate VA's certificate store:

1. Go to the Certificate VA Management menu.
2. In the left frame, select the Manage Certificate Stores link, and then click Add Certificate.

   The Add Certificate form appears.
3. Specify details for storing the certificate:

   - In the Select Certificate Format section, select the "Base 64 encoded (PEM) certificate" option.
   - In the "Select the certificate store to add the certificate" section, select the "CA Delegated Certificates of the VA" option.
4. Click Submit Certificate Type.

   The form for pasting the certificate in its base 64-encoded format shows up.
5. Paste the certificate in the text area of the form. Be sure to include the marker lines, `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`.
6. Click Submit Certificate.

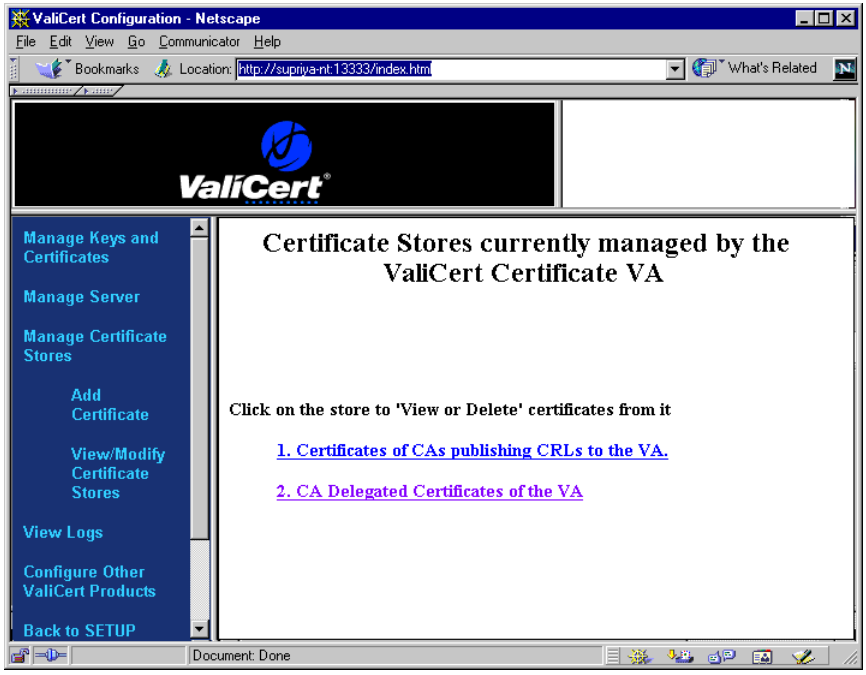
   A page appears indicating whether the certificate is added successfully.

Step E. Verify That the Certificates Are Stored

To verify that the OCSP responder certificate and the CA certificate have been added correctly to the certificate store:

1. In the left frame, click the Manage Certificate Stores link, and then click the View/Modify Certificate Stores option.

   The "Certificate Stores currently managed by the ValiCert Certificate VA" form appears.



2. Click the "Certificates of CAs publishing CRLs to the VA" option.

   The subject name of the Certificate Manager's CA signing certificate you added should appear as a link there.

3. Click the link and make sure the that you added the correct certificate.

4. Next, click the "CA Delegated Certificates of the VA" option.

   The subject name of the OCSP responder certificate you added should appear as a link there.

5. Click the link and make sure the that you added the correct certificate.

Step 6. Restart Certificate VA

For all the changes to take effect, restart Certificate VA.

Step 7. Configure the Certificate Manager for Publishing CRLs

To configure a Certificate Manager to publish CRLs to Certificate VA, follow these steps:

- Step A. Create a Publisher for the CRL
- Step B. Create a Publishing Rule for the CRL
- Step C. Specify CRL Details
- Step D. Set CRL Extensions
- Step E. Make Sure Publishing is Enabled

Step A. Create a Publisher for the CRL

Creating a publisher for the CRL involves creating an instance of the publisher module that enables the Certificate Manager to publish CRLs to Certificate VA. In the next step, when creating the publishing rule for the CRL, you specify the publisher you create here.

To create a publisher:

1. If you closed the CMS window for the Certificate Manager, log in to it again (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.

3. In the navigation tree, select Certificate Manager, select Publishing, and then select Publishers.

   The right pane shows the Publishers Management tab, which lists configured publishers.

4. Click Add.

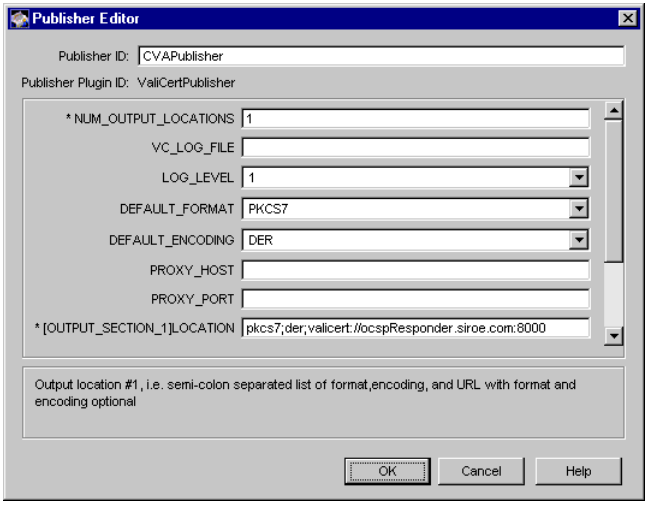
   The Select Publisher Plugin Implementation window appears. It lists registered publisher modules.

5. Select the module named `ValiCertPublisher`.

   Only this publisher module enable the Certificate Manager to publish CRLs to Certificate VA; for details about the module, see "ValiCert Publisher" on page 759.

6. Click Next.

   The Publisher Editor window appears.



7. Enter the appropriate information:

   Publisher ID. type a name for the publisher that will help you identify it later; be sure to use an alphanumeric string with no spaces.

   NUM_OUTPUT_LOCATIONS. Type the total number of output locations. If you leave the field blank, it defaults to 1.

VC_LOG_FILE. Type the path, including the filename, to the file to which you want the server to write (log) messages. If you leave the value field blank, the server logs messages to a file named `./vpublish.log`.

LOG_LEVEL. Select the appropriate level for log messages.

DEFAULT_FORMAT. Select `PKCS7`. (The server considers the value specified in this field when the default format for publishing the CRL is not specified in the `OUTPUT_SECTION` parameter.)

DEFAULT_ENCODING. Select `DER`. (The server considers the value specified in this field when the default encoding is not specified in the `OUTPUT_SECTION` parameter.)

PROXY_HOST. Type the name of the proxy server, if you are using one. If you leave the field blank, no proxy server will be used.

PROXY_PORT. Type the port number of the proxy server, if you are using one. If you leave the field blank, no proxy server will be used.

[OUTPUT_SECTIONS_*]LOCATION. Type the location to publish the CRL. The syntax for specifying the location information must be:

`[pkcs7;][der;]valicert://<host>[:<port>][/<location>]`

Substitute `<host>` with a fully qualified host name of Certificate VA and `<port>` with the service port of Certificate VA (by default it is 80); attributes enclosed within the square bracket are optional.

For example, the value you enter may look like this:
`pkcs7;der;valicert://ocspResponder.siroe.com:8000`.

8. Click OK.

   You are returned to the Publishers Management tab. It should now list the publisher you just created.

Step B. Create a Publishing Rule for the CRL

Creating a publishing rule for the CRL involves creating a rule that uses the publisher you created in “Step A. Create a Publisher for the CRL” on page 876.

To create a publishing rule:

1. In the navigation tree, under Publishing, select Rules.

   The right pane shows the Rules Management tab, which lists configured publishing rules.

2. Click Add.

   The Select Rule Plugin Implementation window appears. It lists registered modules that enable creating of publishing rules.

3. Select the module named `Rule`.

   This is the default module. (If you have registered any custom modules, they too will be available for selection.)

4. Click Next.

   The Rule Editor window appears.

5. Enter the appropriate information:

   Rule ID. Type a unique name for the rule; use an alphanumeric string with no spaces.

   enable. Select this option.

   predicate. Leave this field blank.

   type. Select `crl`.

   mapper. Select `<NONE>`.

   publisher. Select the publisher you created for publishing CRLs to Certificate VA.

6. Click OK.

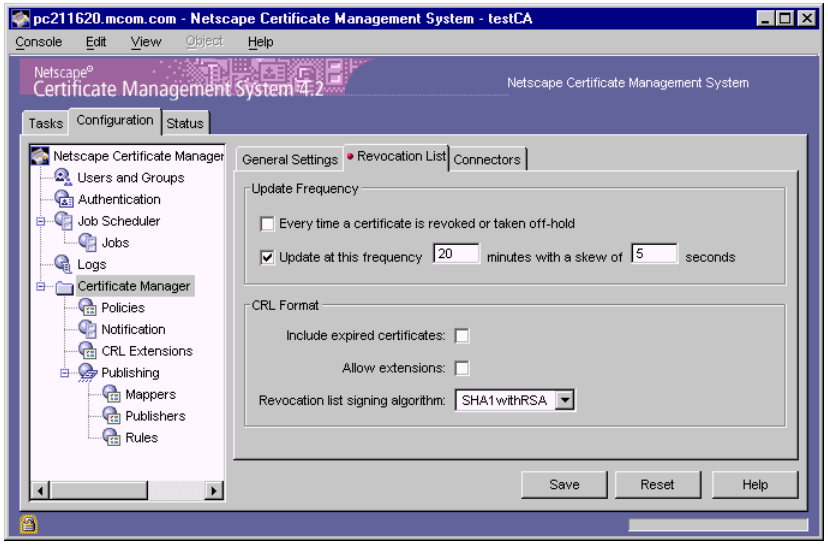
   The Rules Management tab appears, listing the new rule.

Step C. Specify CRL Details

You can specify information, such as the publishing interval, the CRL version (whether to include CRL extensions), and the signing algorithm the Certificate Manager should use for signing the CRL object.

To specify CRL details:

1. Go to the CMS window.
2. In the navigation tree, select Certificate Manager, and then in the right pane, select the Revocation List tab.



3. In the Update Frequency section, specify the interval for publishing the CRL to the directory:

   Every time a certificate is revoked, or taken off-hold. Select this option if you want the Certificate Manager to generate the CRL every time it revokes a certificate. Keep in mind that the Certificate Manager attempts to publish the CRL to the configured directory whenever the CRL is generated, in this case, every time a certificate is revoked. Publishing a CRL can be time consuming if the CRL is large. Configuring the Certificate Manager to publish CRLs every time a certificate is revoked may engage the server for a considerable amount of time; during this time, the Certificate Manager will not be able to service any requests it receives and will not be able to update the directory with any changes it receives.

   (This setting is not recommended for a standard installation. You can select this option if you want to see the results of revocation immediately, for example, when testing whether the server publishes the CRL to the OCSP responder.)

Update at this frequency. Select this option if you want the Certificate Manager to generate CRLs at regular intervals. In this case, the server publishes the CRL to the configured directory at the interval you specify.

In the adjoining text field, type the interval, in minutes, at which the Certificate Manager should publish CRLs. For example, if you want the server to publish CRLs every day, you should type 1440 in this field.

with a skew of. If you configure the server to update the CRL automatically every time period, the server by default adds a 5 second skew to the next update time to allow time to create the CRL and publish it. For example, if you configure the server to update the CRL every 20 minutes, and if the CRL is updated at 16:00:00, the CRL will be updated again at 16:19:55. You can change the skew by manually editing the default value, which is specified in seconds.

4. In the CRL Format section, specify the format for publishing the CRL:

   Include expired certificates. Select this box if you want the server to include revoked certificates that have expired in the CRL.

   Allow extensions. Select this box if you want to allow extensions in the CRL. If you enable this option, the server generates and publishes CRLs conforming to X.509 version 2 standard. If you disable this option, the server generates and publishes CRLs conforming to X.509 version 1 standard. By default, the server publishes version 1 CRLs. If you enable this option, be sure to set the required CRL extensions in "Step D. Set CRL Extensions" on page 882.

   Revocation list signing algorithm. Select the algorithm the server should use to sign the CRL. If the Certificate Manager's signing key type is RSA, select `MD2 with RSA`, `MD5 with RSA`, or `SHA-1 with RSA`. If the Certificate Manager's signing key type is DSA, select `SHA-1 with DSA`.

5. To save your changes, click Save.

   The configuration is modified. If the changes you made require you to restart the server, you are prompted accordingly. Don't restart the server yet; you can do this after you've made all the changes.

Step D. Set CRL Extensions

You should go through this step only if you configured the Certificate Manager to publish version 2 CRLs—that is, you selected the "Allow extensions" option in "Step C. Specify CRL Details" on page 879.

During installation, the Certificate Manager creates default CRL extension rules; these are listed in Table 21.10 on page 766. Note that the server is configured to add the CRL Reason extension only; all the other rules are in the *disabled* state. In this step, you modify the default rules to suit your organization's requirements.

To specify the CRL extensions the Certificate Manager should set:

1. In the navigation tree, select Certificate Manager, and then select CRL Extensions.

   The right pane shows the CRL Extensions Management tab, which lists configured extensions.

2. To modify a rule, select it and then click Edit/View.

3. Change the information as appropriate.

   Be sure to supply all the required values. Click the Help button for detailed information on individual parameters.

4. Click OK.

   You are returned to the CRL Extensions Management tab.

5. To modify other rules, repeat steps 2 through 4.

6. Click Refresh to see the updated status of all the rules.

Step E. Make Sure Publishing is Enabled

To make sure that the Certificate Manager is configured for publishing:

1. In the navigation tree, select Certificate Manager, then select Publishing.

   The right pane shows the publishing details necessary for the server to publish to an LDAP-compliant directory, to flat files, or to an online validation authority.

2. Make sure that the Enable Publishing option is selected. If it is already selected, leave it as it is. If it isn't, select it.

   (Leave the "Enable default LDAP connection" option as it is; it specifies that the Certificate Manager is configured to publish certificates and CRLs to an LDAP directory.)

3. If you changed anything, click Save to save the changes.

   If the changes you made require you to restart the server, you are prompted accordingly. In that case, restart the server.

Step 8. Test Publishing

To test whether the Certificate Manager is publishing to Certificate VA properly and to test that the online validation of certificates is taking place, follow these steps:

- Step A. Turn On Revocation Checking
- Step B. Request a Certificate
- Step C. Approve the Request
- Step D. Download the Certificate to the Browser

- Step E. Verify the Certificate in the Browser
- Step F. Check the Certificate VA Status
- Step G. Revoke the Certificate
- Step H. Verify the Certificate in the Client
- Step I. Check the Certificate VA Status Again

Step A. Turn On Revocation Checking

To ensure that Personal Security Manager (the OCSP-compliant client) is configured to verify the revocation status of certificates using the OCSP protocol:

1. Open a web browser window.
2. Open the Personal Security Manager interface.

   In Communicator version 4.7, you can open this window by selecting Communicator from the main menu, selecting Tools, and then selecting Security Info.
3. Select the Advanced tab, and then in left pane, select Options.
4. Click the OCSP Settings button.

   The OCSP Setting window appears.
5. Select the "Use OCSP to verify only certificates that specify an OCSP service URL" option, and click OK.
6. Click on the Close button.

Step B. Request a Certificate

The steps outlined below explain how to request a personal certificate from the Certificate Manager using the manual enrollment method. If you've configured the Certificate Manager for automated certificate issuance, for example for directory-based enrollment, you may use the appropriate form and request a certificate.

To request a client or personal certificate from the Certificate Manager:

1. Go to the end-entity interface of the Certificate Manager you configured (or to the Registration Manager that's connected to this Certificate Manager). The URL is in this form:

   `https://<host_name>:<end_entity_HTTPS_port>`

   or

   `http://<host_name>:<end_entity_HTTP_port>`

2. In the left frame, under Browser, click Manual.

   This opens the manual enrollment form.

3. Fill in all the values and submit the request.

   The client prompts you to enter the password for your key database.

4. When you enter the correct password, the client generates the key pairs.

   Do not interrupt the key-generation process.

Step C. Approve the Request

Skip this step if you requested the certificate using any of the automated enrollment methods. Complete this step if you used the manual enrollment form for requesting the certificate; the request you submitted is waiting in the agent queue for approval by an agent.

To approve the request:

1. Go to the Certificate Manager's Agent Services interface.

   The URL is in this format: `https://<host_name>:<agent_port>`

2. In the left frame, click List Requests.

3. In the form that appears, select the "Show pending requests" option and click Find.

4. In the list of pending requests, identify the request you submitted and approve the request.

   You should see a confirmation page indicating that the certificate has been issued. Don't close the page until after you complete the next step.

Step D. Download the Certificate to the Browser

To download the certificate into the certificate database of Personal Security Manager:

1. In the confirmation page, scroll down to the section that says "Installing this certificate in a client."
2. Check the certificate details for the required extensions.
3. Follow the on-screen instructions and download the certificate to your browser's certificate database.

   (An alternative way to download the certificate is from the Retrieval tab of the end-entity services interface.)

Step E. Verify the Certificate in the Browser

To verify that the certificate has been downloaded into the certificate database of Personal Security Manager:

1. In the browser, open the Personal Security Manager interface.

   In Communicator version 4.7, you can open this window by selecting Communicator from the main menu, selecting Tools, and then selecting Security Info.
2. Click the Certificates tab and then click Mine.

   You should see the name of the certificate you just downloaded.
3. Select the certificate name and click View.

   In the View Security Certificate dialog box that appears, look for a message that says that the certificate is verified.

Step F. Check the Certificate VA Status

To go to the Certificate VA's status page and verify the number of requests Certificate VA has processed so far:

1. Go to the web browser window.
2. Enter this URL: `http://<host_name>:<admin_port>/~stats`

substituting `<hostname>` with the fully qualified host name of Certificate VA and `<admin_port>` with its administration port number.

The status page for Certificate VA appears. This page summarizes the Certificate VA's activity since it was last started.

3. Note values assigned to the "Number of CRLs received" and "Number of certificates checked" fields; they should be zero.

Step G. Revoke the Certificate

To revoke the certificate you issued so that the Certificate Manager publishes the CRL to Certificate VA:

1. Go to the end-entity interface for the Certificate Manager you configured (or to the Registration Manager that's connected to this Certificate Manager). Be sure to go to the HTTPS interface. The URL is in this form:
   `https://<host_name>:<end_entity_HTTPS_port>`
2. Select the Revocation tab.
3. In the left frame, click User Certificate.

   The User Certificate Revocation form appears.
4. In the Revocation Reason section, select Unspecified and click Submit.

   The client shows the "Select a Certificate" dialog box and prompts you to choose the certificate you want to revoke.
5. Select the certificate you downloaded and click OK.

   The Certificate Manager revokes the certificate, constructs the CRL, and publishes the CRL to Certificate VA.

Step H. Verify the Certificate in the Client

To verify that the certificate has been revoked:

1. Open the Personal Security Manager interface.
2. Select the Certificates tab and then click Mine.
3. Select the certificate you revoked and click View.

In the View Security Certificate dialog box that appears, look for a message that says that the certificate could not be verified.

Step I. Check the Certificate VA Status Again

You check Certificate VA status again to verify that these things happened:

- The Certificate Manager published the CRL (the revoked certificate) to Certificate VA.
- The browser sent an OCSP response to Certificate VA (this response was initiated when you clicked the View button).
- Certificate VA sent an OCSP response to the browser.
- The browser used that response to validate the certificate and informed you of its status (that the certificate could not be verified).

To check the Certificate VA status for verification:

1. Go to the Certificate VA's status page.
2. Reload the page (hold down the Shift key and click on the browser's Reload icon.)
3. Compare the information to the one you noted in Step F.

   Note the updated statistics. It should indicate that Personal Security Manager queried the Certificate VA about the status of the certificate and in response, the Certificate VA sent a status.

Managing Mapper and Publisher Modules

This section explains how to use the CMS window to do the following:

- Registering a Mapper or Publisher Module
- Deleting a Mapper or Publisher Module

For information on adding or changing publishing-specific information in the configuration file, see "Changing the Configuration by Editing the Configuration File" on page 86.

Registering a Mapper or Publisher Module

You can register new mapper or publisher plug-in modules in a Certificate Manager's publishing framework. Registering a new mapper or publisher module involves specifying the name of the module and the full name of the Java class that implements the mapper or publisher interface. For example, you can add a mapper implementation, named as follows, to the Certificate Manager's policy framework:

```
com.netscape.publishing.customMapper
```

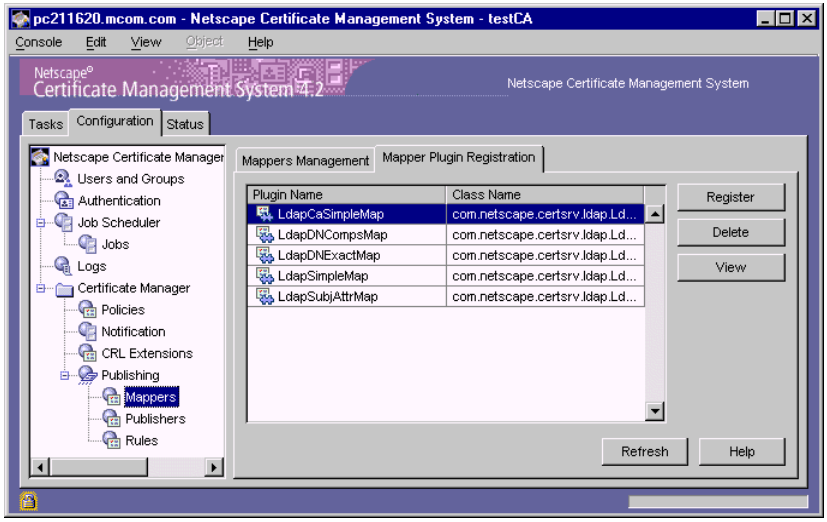
Before registering a plug-in module, be sure to put the Java class for the module in the `classes` directory (the implementation must be on the class path).

To register a mapper or publisher module in a Certificate Manager's publishing framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Certificate Manager, and then select Publishing.
4. Select the appropriate object under Publishing:
   - To register a mapper module, select Mappers, and then in the right pane, select the Mapper Plugin Registration tab.

- To register a publisher module, select Publishers, and then in the right pane, select the Publisher Plugin Registration tab.

This tab lists registered plug-in modules.



5. Click Register.

   If you selected Mapper, the Register Mapper Plugin Implementation window appears. If you selected Publisher, the Register Publisher Plugin Implementation window appears.

6. Specify information as appropriate:

   Plugin name. Type the name of the plug-in module.

   Class name. Type the full name of the class for this module—that is, the path to the implementing Java class. If this class is part of a package, be sure to include the package name. For example, if you are registering a class named `myMapper` and if this class is in a package named `com.myCompany`, type `com.myCompany.myMapper`.

7. Click OK.

   You are returned to the Mapper Plugin Registration tab or Publisher Plugin Registration tab.

8. To view the updated configuration, click Refresh.

Deleting a Mapper or Publisher Module

You can delete unwanted mapper or publisher plug-in modules using the CMS window. Before deleting a module, be sure to delete all the publishing rules that are based on this module.

To delete a mapper or publisher module from a Certificate Manager's publishing framework:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Configuration tab.
3. In the navigation tree, select Certificate Manager, and then select Publishing.
   - To delete a mapper module, select Mappers, and then in the right pane, select the Mapper Plugin Registration tab.
   - To delete a publisher module, select Publishers, and then in the right pane, select the Publisher Plugin Registration tab.

   This tab lists registered plug-in modules.
4. In the Plugin Name list, select the module you want to delete and click Delete.
5. When prompted, confirm the delete action.

8

Agent and End-Entity Interfaces

Chapter

23

Introduction to End-Entity and Agent Interfaces

Netscape Certificate Management System (CMS) provides HTML forms-based interfaces for agents and end entities to use in performing certificate- and key-related operations. This chapter introduces these forms and explains how they work. You can use the forms as they are provided out of the box or customize them to meet your organization's requirements.

This chapter has the following sections:

For details on customizing these forms, see "Customizing End-Entity and Agent Interfaces" on page 905.

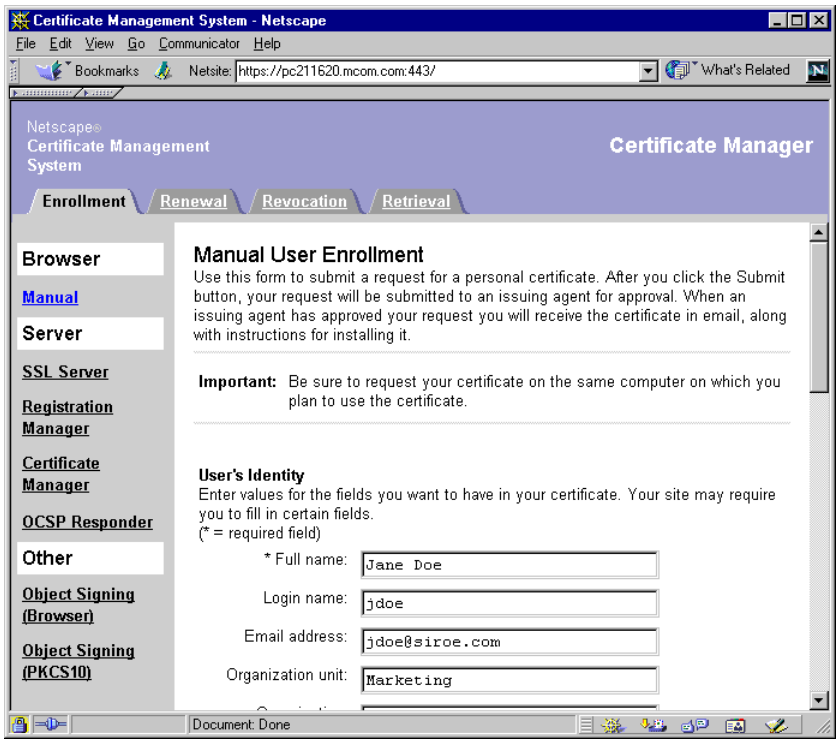
End-Entity Services

Certificate Management System provides HTML forms for the various entities—people, routers, servers, and others—that use certificates to identify themselves and that need to be able to request certificate issuance and management operations. These forms, collectively called the *end-entity services* interface, use different protocols and life-cycle management procedures for different kinds of end entities. For example, the Certificate Manager provides separate certificate enrollment forms for clients such as Netscape Navigator 3.x, versions of

Netscape Communicator later than 4.5, and Microsoft Internet Explorer. The reason for this is that end entities running Navigator 3.x and Communicator versions earlier than 4.5 present an enrollment form based on the use of the HTML tag KEYGEN to generate keys; end entities running Internet Explorer present a form based on PKCS #10, the RSA standard for certificate request syntax.

Figure 23.1 shows the end-entity services interface hosted by a Certificate Manager.

Figure 23.1 End-entity services interface



For a summary of the various end entities, protocols, cryptographic algorithms, and key pairs (single or dual) supported by Certificate Management System, see Table 23.1 on page 898.

For a complete list of the end-entity forms—for enrollment, renewal, retrieval, revocation, and key recovery—that come with Certificate Management System, see "End-Entity Forms and Templates" on page 913.

How Client Type Determines the End-Entity Interface

Each type of end-entity form provided by Certificate Management System is served by a servlet. This servlet determines which version of the form to present based on information about the end entity (the type, version, language, and so on), information in the form itself, and other factors.

Each form also specifies both an authentication manager and an output template:

- An authentication manager is a configured instance of an authentication plug-in module. When Certificate Management System receives a request from an end entity, it uses the authentication manager specified by the request to determine how to authenticate the end entity. For more information, see "Setting Up Authentication for End-User Enrollment" on page 387.
- The output template is an HTML page with embedded JavaScript used to return information from the end entity to the servlet. For more information, see "Responses and Output Templates" on page 908.

Based on all the information, a form's servlet sends the end entity the version of the form (including the embedded JavaScript code) appropriate for that end entity. For example, in the case of end entities that support the `KEYGEN` tag, the Certificate Manager or Registration Manager sends a form that uses `KEYGEN` to generate keys and formulate a certificate request. In the case of end entities that support the Certificate Management Message Format (CMMF) protocol, the Certificate Manager or Registration Manager sends a form that uses a JavaScript API to fully automate both key generation and certificate issuance.

Certificate Request Formats Specific to End Entities

Table 23.1 lists the forms provided by the Certificate Manager and Registration Manager for certificate issuance and life-cycle management operations, and indicates supported authentication mechanisms and request formats.

You can customize any of the default forms and their corresponding servlets and output templates. For details, see "Customizing End-Entity and Agent Interfaces" on page 905.

Table 23.1 Summary of end-entity forms, authentication methods and certificate request formats

Form for end-entity operation	Authentication method	Supported certificate request formats
Certificate enrollment		
Client (end user) certificates	Manual, LDAP directory based, and NIS server based	• KEYGEN for Navigator/Communicator • PKCS #10 for Internet Explorer • Certificate Request Message Format (CRMF) for future versions of Communicator
Server certificates	Manual	PKCS #10
Cisco routers	Manual or automated	Certificate Enrollment protocol (CEP)
Certificate renewal		
Client (end user) certificates	SSL client authentication	• KEYGEN for Navigator/Communicator • PKCS #10 for Internet Explorer • CRMF for future versions of Communicator
Server certificates	Manual	PKCS #10
Cisco routers	Manual	CEP
Certificate revocation		
Client (end user) certificates	SSL client authentication and challenge-password based	• KEYGEN for Navigator/Communicator • PKCS #10 for IE • CRMF for future versions of Communicator
Server certificates	Manual	PKCS #10
Cisco routers	Manual	CEP

Table 23.1 Summary of end-entity forms, authentication methods and certificate request formats (Continued)

Form for end-entity operation	Authentication method	Supported certificate request formats
Encryption private key storage and recovery		
Client (end user) certificates	Not applicable	• Not supported for clients that can't generate dual key pairs • CRMF for future versions of Communicator

Agent Services

As an administrator, you can designate privileged users, called agents, for each subsystem. Agents are responsible for the day-to-day operation of requests from end entities. To enable agents to accomplish their duties, Certificate Management System provides a set of HTML forms for Certificate Manager, Registration Manager, and Data Recovery Manager agents. Collectively, these forms are called the *Agent Services* interface.

Depending on the choices you made during installation, a combination of the following agent services will be installed:

- Certificate Manager Agent Services
- Registration Manager Agent Services
- Data Recovery Manager Agent Services

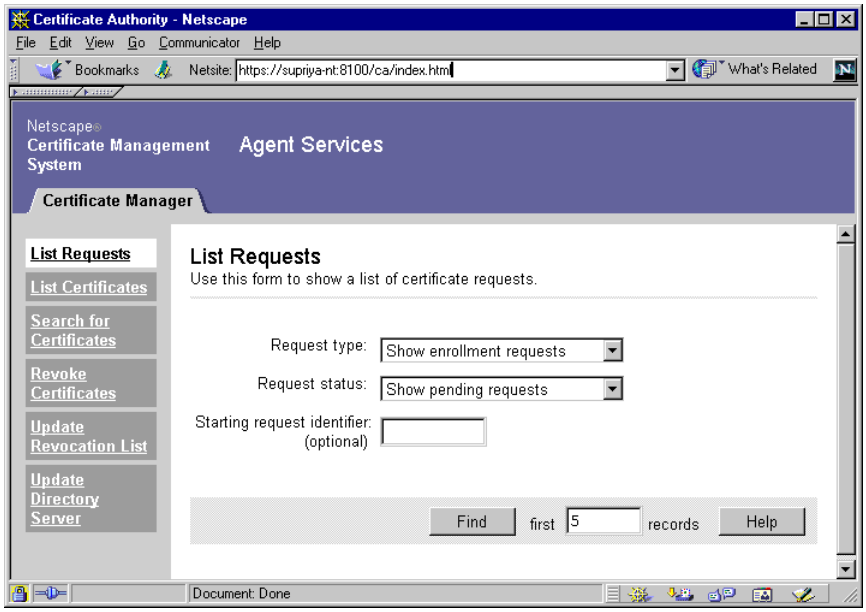
This section gives an overview of these forms and explains how to access them. For a complete list of the agent forms and output templates that come with Certificate Management System, see "Agent Forms and Templates" on page 921. For step-by-step instructions on using the agent forms, see *Netscape Certificate Management System Agent's Guide*. For information on locating this guide, see "Where to Go for Related Information" on page 39.

Note that accessing the Agent Services interface is a privileged operation, requiring certificate-based (or *strong*) authentication. It can be done only by users belonging to authorized agent groups maintained by Certificate Management System in its internal database. For details, see "Agents" on page 173.

Certificate Manager Agent Services

The Certificate Manager Agent Services interface enables a Certificate Manager agent to interact with the Certificate Manager (the server). Figure 23.2 shows the Certificate Manager Agent Services interface.

Figure 23.2 Certificate Manager Agent Services interface



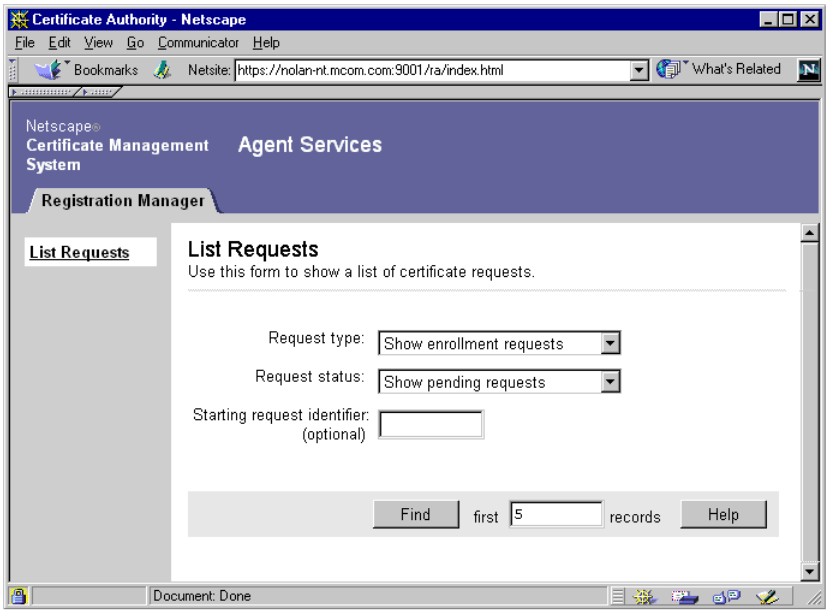
Using the default forms, a Certificate Manager agent can accomplish tasks such as these:

- Listing *deferred* certificate requests from end entities and process them
- Listing certificates issued by the server
- Searching for certificates issued by the server
- Revoking certificates issued by the server
- Updating certificates and certificate revocation lists (CRLs) maintained in the publishing directory

Registration Manager Agent Services

The Registration Manager Agent Services interface enables a Registration Manager agent to interact with the Registration Manager (the server). Figure 23.3 shows the Registration Manager Agent Services interface.

Figure 23.3 Registration Manager Agent Services interface

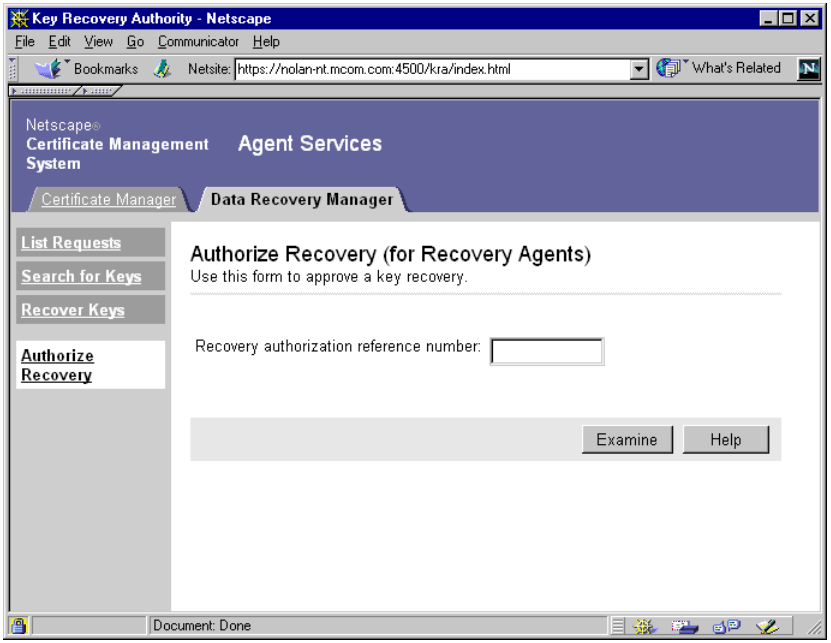


Using the default forms, a Registration Manager agent can list *deferred* certificate requests from end entities and process them.

Data Recovery Manager Agent Services

The Data Recovery Manager Agent Services interface enables a Data Recovery Manager agent to interact with the Data Recovery Manager (the server). Figure 23.4 shows the Data Recovery Manager Agent Services interface.

Figure 23.4 Data Recovery Manager Agent Services interface



Using the default forms, a Data Recovery Manager agent can search for and recover end users' encryption private keys from the key archive. (Key recovery requires authorization from key recovery agents; see "Key Recovery Process" on page 1122.)

Accessing the Agent Services Interface

Access to the Agent Services interface is restricted to authorized agents only. For details, see "Agents" on page 173.

To access the Agent Services interface for a particular subsystem:

1. Open a web browser.
2. Go to the page where the Agent Services interface for Certificate Management System is installed.

The default URL for this page is:

`https://<host_name>:<agent_port>`

`<host_name>` is in the form `<machine_name>.<your_domain>.<domain>`

If you have customized Certificate Management System, go to the page containing the agent forms that you would use to submit a request.

3. In the Agent Services menu, choose the agent services you require:

   - To access the agent services for the Certificate Manager, click the Certificate Manager Agent Services link.
   - To access the agent services for the Registration Manager, click the Registration Manager Agent Services link.
   - To access the agent services for the Data Recovery Manager, click the Data Recovery Manager Agent Services link.

   The appropriate interface appears.

Chapter

24

Customizing End-Entity and Agent Interfaces

The services interfaces that come with Netscape Certificate Management System (CMS) make it possible for end-entities and agents to interact with the server. Your end-entities and agents can use the interface's HTML-based forms to carry out various certificate and key-related operations, such as enrolling for, renewing, and revoking certificates.

You can use the default forms as they are, customize them, or develop your own forms to suit your organization's policies or terminology. This chapter explains how to customize the forms and templates used by the interfaces.

The chapter has the following sections:

What You Need to Know to Change Forms

Changing the default forms' appearance requires only a basic knowledge of HTML and a text editor. However, more significant customizing efforts require a good understanding of the following:

- HTTP, Query URLs, and HTML Forms
- JavaScript
- The way the End-Entity Services interface works

HTTP, Query URLs, and HTML Forms

Requests from the end-entity services interface to Certificate Management System are submitted using the HTTP `GET` and `POST` methods. Requests take the form of query URLs (in the case of the `GET` method) or data sent through standard output (in the case of the `POST` method).

For background on these topics, consult books on authoring for the World Wide Web, on HTML, and on the HTML specification. Another good source is Netscape's documentation on creating Internet sites; that documentation is available at this URL:

`http://home.netscape.com/assist/net_sites/index.html`

JavaScript

JavaScript is a scripting language that most browser software, including Netscape Navigator and Communicator, used for dynamic forms. (JavaScript is not the same as the more sophisticated and powerful Java language that is also supported by Netscape clients.)

In the agent and end-entity services forms, JavaScript is used to check input values and to formulate requests to the CMS server. JavaScript is also used in the output templates to present and format responses from the CMS server.

To customize the forms and templates, you need to be familiar with JavaScript. For more information on JavaScript and its use in Netscape browsers, see the *Netscape JavaScript Authoring Guide* available at this URL:

```
http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/
index.html
```

There are also several books on this topic.

How the Forms Work

Administrators, end-entities, and agents request service operations using the HTTP or HTTPS (HTTP over SSL) protocol using either the `GET` method (by submitting query URLs) or the `POST` method (by submitting a URL-encoded form with the content-type `application/x-www-form-urlencoded`). The `GET` method and the `POST` method result in a set of name-value pairs; this set constitutes the request.

For certificate service operations, the URI portion should indicate

```
/<operation>
```

where `operation` designates the certificate (management) service portion, such as enrollment, retrieval, renewal, or revocation of the CMS server. Any HTTP operations with URIs that do not begin with the `/<operation>` prefix are treated as requests for other kinds of web service by the CMS server. See the sections "End-entity Interface Reference" on page 927 and "Agent Interface Reference" on page 972 for details on all the available operations.

Requests Sent to the CMS server

The services interface handles a set of operations. Each operation has a name and expects a specific set of parameters.

(The examples in this chapter are what you would see if Certificate Management System were running on the host `certs.mydom.com` and listening on the standard HTTPS port.)

For example, the `displayBySerial` interface displays the certificate with the serial number matching the `serialNumber` parameter. To use the `displayBySerial` interface to retrieve the certificate with serial number 58 (0x3a) using HTTP GET, you would use the following URL:

```
https://certs.mydom.com/displayBySerial?serialNumber=58
```

You could embed this URL in a link on an HTML page to allow users to view this certificate.

If you used an HTML form (rather than a query URL) to invoke this operation, the HTML for the form might look like this:

```
<FORM ACTION="https://certs.mydom.com/displayBySerial"
METHOD="POST">
Serial Number: <INPUT TYPE="TEXT" NAME="serialNumber">
<INPUT TYPE="SUBMIT" VALUE="Display This Certificate">
</FORM>
```

In the resulting form, the user enters the serial number of the certificate to be displayed.

Responses and Output Templates

Certificate Management System responds to service requests by sending back an HTML page built from two parts: a fragment of JavaScript code containing the data resulting from the operation and a template defining how the data is processed and displayed.

The fragment of JavaScript code consists of a `result` object that contains data properties only (no methods). The properties of the object correspond to parts of the response.

The template generally contains a combination of HTML and JavaScript code that processes and displays data. The template is set up to make use of the data in the `result` object.

In responding to a request, Certificate Management System determines the data that needs to be returned, embeds the data in the definition of the `result` object, and inserts the `result` object in the template. When the browser receives the constructed HTML page from the CMS server, the JavaScript code in the template file looks at the values in the `result` object and uses the data to display the HTML page to the user.

Because the functions that manipulate and display the data are accessible to you in the plain-text template files (as opposed to being hardcoded in the CMS server's libraries), you can customize the way data is used and presented to the user by editing the JavaScript and HTML in the template files.

For example, the `displayBySerial` operation generates the following JavaScript code fragment:

```
<SCRIPT LANGUAGE="JavaScript">
var header = new Object();
var fixed = new Object();
var recordSet = new Array;
var result = new Object();
var httpParamsCount = 0;
var httpHeadersCount = 0;
var authTokenCount = 0;
var serverAttrsCount = 0;
header.HTTP_PARAMS = new Array;
header.HTTP_HEADERS = new Array;
header.AUTH_TOKEN = new Array;
header.SERVER_ATTRS = new Array;
header.certPrettyPrint = [long string containing pretty-
printed certificate]
header.noCertImport = false;
header.certFingerprint = [string containing certificate
fingerprints]
header.authorityid = "ca";
header.serialNumber = 5;
header.emailCert = true;
header.certChainBase64 = [string containing base-64 encoded
certificate]
result.header = header;
result.fixed = fixed;
result.recordSet = recordSet;
</SCRIPT>
```

Notice how this code fragment defines an object named `result` and puts the resulting data from the operation in the properties of that object. Each certificate service operation returns an object named `result`. The contents of the `result` object are specific to the operation.

When it responds to the request, the `displayBySerial` interface running on Certificate Management System inserts this JavaScript fragment into the template file it uses to return results to the requestor. the CMS server inserts the fragment in the template file where it finds the tag `<CMS_TEMPLATE>`. It then returns the

template with the inserted fragment to the client. The client then processes the completed template and displays the resulting page. In the case of the `displayBySerial` operation, the template file uses JavaScript and HTML to display the contents of the `result` object to the user.

Because the data from the operation is available in the `result` object, you can customize the JavaScript in the template or write your own functions to use this data. For example, to access the certificate's serial number, you can write a JavaScript function that uses `result.header.serialNumber`.

Templates for each operation are stored in the `web` subdirectory of the CMS server instance. The `web` subdirectory contains the following subdirectories where forms and templates are located:

- `ee` for end-entity interfaces
- `agent/ca` for Certificate Manager agent interfaces
- `agent/kra` for Data Recovery Manager agent interfaces
- `agent/ra` for Registration Manager agent interfaces

the CMS server reads the templates dynamically; you do not have to restart the CMS server for it to read changes to the template files.

Errors and the Error Template

All certificate service errors in the end-entity interface are returned through a single template called `GenError.template`. The `error result` object contains the following data properties:

```
<SCRIPT LANGUAGE="JavaScript">
var header = new Object();
var result = new Object();
header.errorDetails = [a string describing the context of the
error]
header.errorDescription = [a string describing the error]
result.header = header;
</SCRIPT>
```

The default CMS error template prints the information in the `error result` object along with some explanatory text.

Displaying Forms in Non-English Languages

The forms and response templates that come with Certificate Management System are all in English. Certificate Management System supports forms and templates in other languages, and multiple languages can be supported on the same CMS server instance. Every aspect of the CMS server is designed to accomodate multiple languages, including all storage and certificate processing (it is possible to have certificate subject names with data in Chinese, for example). The CMS administration console windows support data in non-English languages, but the messages and menu items cannot be localized.

When an HTTP or HTTPS request arrives at the CMS server, the CMS server checks the HTTP `Accept-language` header to see what languages are preferred by the requestor. For example, a client that prefers content in Korean would have the value "ko" in the `Accept-language` header. The server looks in the directory where the default form would be stored to see if there is a directory matching the first value in the `Accept-language` header. If there is such a directory, the CMS server looks for the correct form or template in the language-specific directory; if the form or template is not found, the default is still used.

For example, the manual user enrollment form is `ManUserEnroll.html`. It is stored in the `web/ee/` directory below the CMS server root. If you wanted to provide a version of this form in French and German for your users, you would translate the form, create the directories `web/ee/fr` (French) and `web/ee/de` (German), and put the translated versions of the form in the appropriate, language-specific subdirectory. The appropriate form is sent to users automatically based on the language preferences set in their browsers.

Localized versions of the agent forms and templates are supported in the same way. Create language-specific subdirectories of `web/agent/ca`, `web/agent/kra`, and `web/agent/ra` to provide forms and templates for agents in non-English languages.

Note that if a browser sends more than one language, the CMS server will try to match one of the browser's language preferences with the default locale of the system where the server is running. If no match is found, the default page in English will be returned to the browser. Users having trouble accessing your localized content should make sure they have only one language set in their browsers.

Certificate Management System uses a default character set for each language (see Table 24.1). If you want to use a different character set for a language, you must edit the CMS server configuration file CMS.cfg and add a line with the following format:

```
i18nCharset.<lang>=<charset>
```

Where <lang> is the two-letter code for the language (the same as the directory where the localized files are stored) and <charset> is the character set to use with files in that language. For example, to use a character set named EUC_KR for Korean-language (ko) content, add the following line to CMS.cfg:

```
i18nCharset.ko=EUC_KR
```

The following table lists the languages supported by Certificate Management System, the two-letter language code to use for language-specific directories, and the default character set Certificate Management System uses for the language:

Table 24.1 Languages and Default Character Sets

Language	Code	Character set	Language	Code	Character set
Albanian	sq	ISO-8859-2	Arabic	ar	ISO-8859-6
Bulgarian	bg	ISO-8859-5	Byelorussian	be	ISO-8859-5
Catalan (Spanish)	ca	ISO-8859-1	Chinese (Simplified/Mainland)	zh	GB2312
Chinese (Traditional/Taiwan)	zh	Big5	Croatian	hr	ISO-8859-2
Czech	cs	ISO-8859-2	Danish	da	ISO-8859-1
Dutch	nl	ISO-8859-1	English	en	ISO-8859-1
Estonian	et	ISO-8859-1	Finnish	fi	ISO-8859-1
French	fr	ISO-8859-1	German	de	ISO-8859-1
Greek	el	ISO-8859-7	Hebrew	he	ISO-8859-8
Hungarian	hu	ISO-8859-2	Icelandic	is	ISO-8859-1
Italian	it	ISO-8859-1	Japanese	ja	Shift_JIS
Korean	ko	KSC_5601	Latvian (Lettish)	lv	ISO-8859-2
Lithuanian	lt	ISO-8859-2	Macedonian	mk	ISO-8859-5
Norwegian	no	ISO-8859-1	Polish	pl	ISO-8859-2

Table 24.1 Languages and Default Character Sets (Continued)

Language	Code	Character set	Language	Code	Character set
Portuguese	pt	ISO-8859-1	Romanian	ro	ISO-8859-2
Russian	ru	ISO-8859-5	Serbian	sr	ISO-8859-5
Serbo-Croatian	sh	ISO-8859-5	Slovak	sk	ISO-8859-2
Slovenian	sl	ISO-8859-2	Spanish	es	ISO-8859-1
Swedish	sv	ISO-8859-1	Turkish	tr	ISO-8859-9
Ukranian	uk	ISO-8859-5			

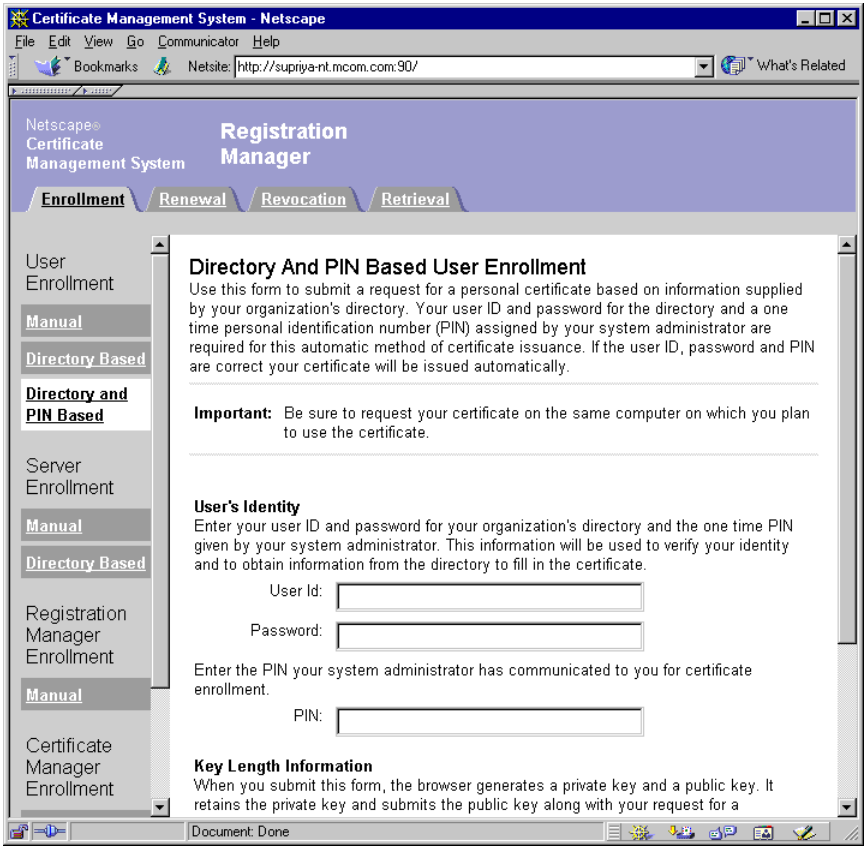
End-Entity Forms and Templates

This section describes the end-entity interface and its default forms.

The end-entity services interface is divided into three parts or frames—top, menu, and content. The top frame includes tabs that are specific to end-entity operations, such as certificate enrollments and renewals. The menu lists all the operations supported by the selected tab. The content shows the form

pertaining to the operation an end entity chooses in the menu; the form contains information to carry out the selected operation. Figure 24.1 shows the end-entity interface of a Registration Manager.

Figure 24.1 End-entity services interface



Locating End-Entity Forms and Templates

You can find the HTML forms and the corresponding output templates for the end-entity interface at this location:

```
<server_root>/cert-<instance_id>/web/ee
```

Forms for Certificate Enrollment

Table 24.2 lists the file names of forms that appear as menu options in the Enrollment tab of the end-entity interface. The forms are available on Certificate Manager instances and Registration Manager instances, The only exception is that the Certificate Manager enrollment form is available only on Certificate Manager instances.

Table 24.2 Forms for end-entity enrollment

Form Type: Menu Link and `Filename`	What form is used for...
User Enrollment (lists menu options for end-user enrollment)	
Manual (`ManUserEnroll.html`)	End users can use the User Enrollment forms to request SSL client and S/MIME certificates. Except for Manual, these links only appear when an appropriate authentication manager has been configured on the CMS server.
Directory Based (`DirUserEnroll.html`)	Enroll using directory user ID and password.
Directory and PIN Based (`DirPinUserEnroll.html`)	Enroll using directory user ID, password, and one time PIN.
NIS Server (`NISUserEnroll.html`)	Enroll using authentication against a NIS server.
Portal (`PortalEnrollment.html`)	Enroll using any unique user ID and a password.
Certificate (`CertBasedDualEnroll.html`)	Enroll for dual key certificates using a pre-issued certificate (on a hardware token) for authentication.
Certificate (`CertBasedSingleEnroll.html`)	Enroll for a single certificate using a pre-issued certificate (on a hardware token) for authentication. (This form is not used in the default interface.)
Certificate (`CertBasedEncryptionEnroll.html`)	Enroll for an encryption certificate only using a pre-issued certificate (on a hardware token) for authentication. (Thisform is not used in the default interface.)
Server Enrollment (lists menu options for server enrollment)	
SSL Server (`ManServerEnroll.html`)	Server administrators can use this form to request SSL server certificates for servers.

Table 24.2 Forms for end-entity enrollment (Continued)

Form Type: Menu Link and `Filename`	What form is used for...
Directory Based Server (`DirServerEnroll.html`)	Server administrators can use this form to request SSL server certificates for servers.
OCSP Responder (`OCSPResponder.html`)	Server administrators can use this form to request signing certificates for OCSP Responder servers.
Registration Manager Enrollment (lists menu options for Registration Manager enrollment)	
Registration Manager (`ManRAEnroll.html`)	Registration Manager administrators can use this form to request a *signing certificate* for a Registration Manager. For details about this certificate, see "Signing Key Pair and Certificate" on page 230.
Certificate Manager Enrollment (lists menu options for Certificate Manager enrollment)	
Certificate Manager (`ManCAEnroll.html`)	Certificate Manager administrators can use this form to request *CA signing certificates* for Certificate Managers functioning as subordinate CAs. For details about the CA signing certificate, see "CA Signing Key Pair and Certificate" on page 226.
Object Signing Enrollment (lists menu options for object signing enrollment)	
Object Signing (Browser) (`ManObjSignEnroll.html`) Object Signing (PKCS10) (`ObjSignPKCS10Enroll.html`)	End users and administrators can use this form to enroll for a certificate that allows them to sign objects, such as Java applets. Both the Certificate Manager and Registration Manager provide this form.

Forms for Certificate Renewal

Table 24.3 lists the forms that correspond to the menu options in the Renewal tab of the end-entity interface on Certificate Manager instances and Registration Manager instances.

Table 24.3 Forms for certificate renewal

Menu Link and Filename	What form is used for...
Server Certificate (ServerRenewal.html)	Server administrators can use this form to renew server certificates.
User Certificate (UserRenewal.html)	End users can use this form to renew their SSL client certificates and their S/MIME certificates if the S/MIME certificates were issued with the SSL client bit set.

Forms for Certificate Revocation

Table 24.4 lists the forms that correspond to the menu options in the Revocation tab of the end-entity services interface.

Table 24.4 Forms for certificate revocation

Menu Link and Filename	What form is used for...
Certificate (challenge phrase-based) (ChallengeRevoke1.html)	End users can use this form to revoke their SSL client certificates using a password created during enrollment.
Server Revocation (ServerRevocation.html)	Server administrators can use this form to revoke server certificates.
User Revocation (UserRevocation.html)	End users can use this form to revoke their SSL client certificates using SSL client authentication.

Forms for Certificate Retrieval

Table 24.5 lists the forms that correspond to the menu options in the Retrieval tab of the end-entity interface on Certificate Manager instances. Only the Import CA Certificate Chain interface is also available on Registration Manager instances.

Table 24.5 Forms provided for certificate retrieval

Menu Link and `Filename`	What form is used for...
List Certificates (`queryBySerial.html`)	End users and administrators can use this form to list certificates based on their serial numbers.
Search for Certificates (`queryCert.html`)	End users and administrators can use this form to search for specific certificates. The search criteria can be a combination of the following: • Serial number of the certificate • Subject name of the certificate • Revocation status of the certificate • Issuing Information—when the certificate was issued • Validity period of the certificate • Type of certificate
Import CA Certificate Chain (`GetCAChain.html`)	End users and administrators can use this form to import the certificate chain of a Certificate Manager (CA) into their browsers or servers. They can • Import the CA certificate chain into their browsers • Download the CA certificate chain in binary form • View the CA certificate chain for importing into a server • Display certificates in the CA certificate chain for importing individually into a server
Import Certificate Revocation List (`DisplayCRL.html`)	End users and administrators can use this form to: • Manually check the revocation status of a particular certificate (if they are not sure whether they have the latest version of the CRL) • Import the latest CRL to Netscape Navigator • Download the latest CRL in binary form • View the CRL header information

Forms for Key Recovery

Table 24.6 lists the form that corresponds to the menu option in the Recovery tab of the end-entity interface. This form is available on a Certificate Manager instance or a Registration Manager instance that is configured as a trusted manager for a Data Recovery Manager instance.

Table 24.6 Form for encryption private key recovery

Menu Link and `Filename`	What form is used for...
Key Recovery (KeyRecovery.html)	End users can use this form to retrieve their encryption private keys from the Data Recovery Manager.

Other Forms

Table 24.6 lists common forms that are used by the operation-specific forms in the end-entity interface.

Table 24.7 Files and forms used by other forms

Form filename	What form is used for...
`enrollMenu.html`	This file loads and highlights the Enrollment tab.
`renewalMenu.html`	This file loads and highlights the Renewal tab.
`recoveryMenu.html`	This file loads and highlights the Recovery tab.
`retrievalMenu.html`	This file loads and highlights the Retrieval tab.
`index.html`	This file contains the menu options. To change the name of an option, search for it in the file and then edit it.
`*.js`	Files with a `.js` file extension include JavaScript helper functions that are used by other forms.
`xenroll.dll`	This file enables the end-user enrollment forms to work with Microsoft Internet Explorer.

Output Templates for End-Entity Interfaces

Table 24.8 lists the default templates that are used by the end-enetity interfaces to return data to the requestor.

Table 24.8 Response templates used by the end-entity interface

Template filename	Description
`displayBySerial.template`	Used to display information pertaining to a certificate when users view an individual certificate (for example, when they click the Details button next to a certificate).
`EnrollSuccess.template`	Used to inform the CMS administrator that the agent certificate he or she requested has been successfully installed in the subsystem's internal database.
`GenError.template`	Used to display error messages to the user.
`GenPending.template`	Used to inform a user requesting a certificate that the request has been queued for agent approval.
`GenRejected.template`	Used to inform a user requesting a certificate that the request has been rejected by the CMS server.
`GenSuccess.template`	Used to inform a user requesting a certificate that the request has been approved by the CMS server.
`GenSvcPending.template`	Used to inform a user requesting a certificate that the request has been queued for agent approval.
`GenUnauthorized.template`	Used to inform users when thay perform unauthorized operations.
`GenUnexpectedError.template`	Used to inform the user that the CMS server encountered an unexpected error while processing the request.
`ImportCert.template`	Used to display the CA certificate when users import the CA certificate.
`queryCert.template`	Used to display the list of certificates when users search for certificates.
`RenewalSuccess.template`	Used to inform a user requesting a certificate renewal that the request has been successfully renewed.
`RevocationSuccess.template`	Used to inform a user requesting a certificate revocation that the certificate has been revoked.

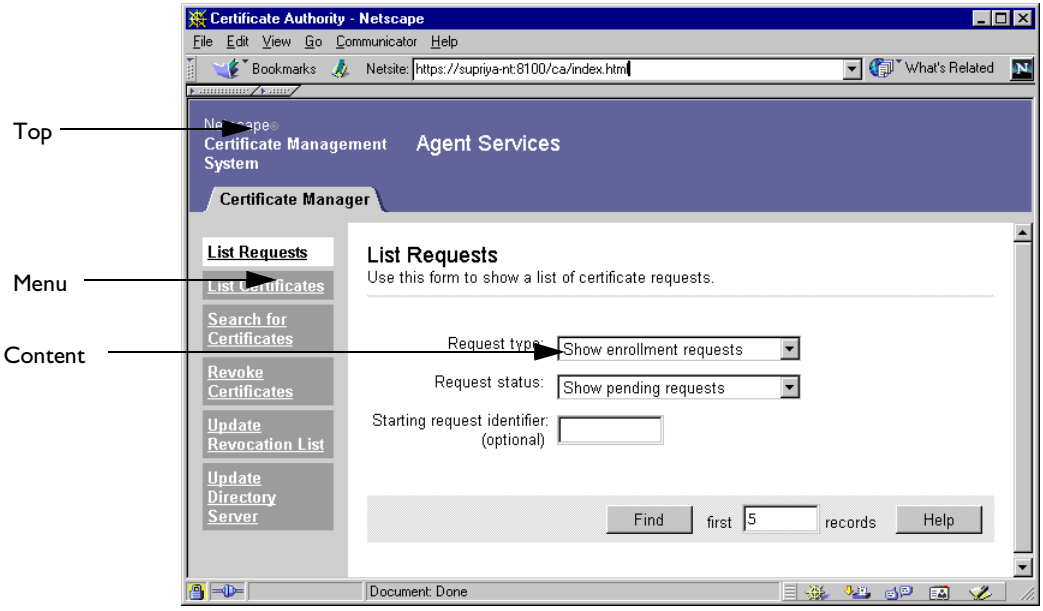
Agent Forms and Templates

This section describes the Agent Services interface, gives the location of the agent forms and output templates, and lists all of the default forms and templates.

Structure of the Agent Services Interface

As shown in Figure 24.2, the Agent Services interface is divided into three parts or frames—top, menu, and content. The top frame includes the tabs that allow you to select a subsystem. The menu lists all the operations supported by the selected subsystem. The content shows the form pertaining to the operation an agent chooses in the menu; the form contains information to carry out the selected operation.

Figure 24.2 Various parts of the Agent Services interface



Locating Agent Forms and Templates

You can find the HTML forms specific to agent operations and the corresponding output templates at this location:

```
<server_root>/cert-<instance_id>/web/agent/<subsystem>
```

`<server_root>` is the directory where the CMS binaries are kept, as specified during installation.

`<instance_id>` is the ID for this instance of Certificate Management System. You specified this ID during installation.

`<subsystem>` refers to the forms directory pertaining to a subsystem, the Certificate Manager (`ca`), Registration Manager (`ra`), or Data Recovery Manager (`kra`).

JavaScript Used By All Interfaces

This section describes the JavaScript variables that are common to all responses from end-entity and agent interfaces. The interface definitions in subsequent sections give details about additional JavaScript that may be added by specific interfaces.

The CMS server handling the interface request replaces the `<CMS_TEMPLATE>` tag in a template with the JavaScript variables described in this section. When you modify or create templates, make sure the `<CMS_TEMPLATE>` tag appears in the file if your response needs to use any of the variables returned by an interface.

The JavaScript included in a response allows you to customize how the response is displayed or processed. For example, the `queryCert.template` file is used to list certificates returned by the List Certificates interface. The template makes extensive use of the JavaScript in the response to display a summary of each certificate and also to create new HTTP forms that can show details about a certificate.

All responses will include the following JavaScript:

```
<SCRIPT LANGUAGE="JavaScript">
var header = new Object();
var fixed = new Object();
```

```
var recordSet = new Array;
var result = new Object();
var httpParamsCount = 0;
var httpHeadersCount = 0;
var authTokenCount = 0;
var serverAttrsCount = 0;
header.HTTP_PARAMS = new Array;
header.HTTP_HEADERS = new Array;
header.AUTH_TOKEN = new Array;
header.SERVER_ATTRS = new Array;
fixed.preserved = "foo";
var recordCount = 0;
var record;
record = new Object;
record.HTTP_PARAMS = new Array;
record.HTTP_HEADERS = new Array;
record.AUTH_TOKEN = new Array;
record.SERVER_ATTRS = new Array;
recordSet[recordCount++] = record;
result.header = header;
result.fixed = fixed;
result.recordSet = recordSet;
</SCRIPT>
```

On its own, the base JavaScript is not very useful. Data pertinent to the response is added to this framework by the interface that creates the response. For example, an interface that lists a certificate might add the base-64 encoding of the certificate to the `record` object.

The `result` object contains most of the useful data, including the `header` object, the `fixed` object, and the `recordSet` array. Responses will usually add relevant data to one of these components of the result object. The purpose of these components can be summarized as follows:

- The `header` object contains variables and data that apply generally to all parts of the response. For example, a response including several certificates would store the total number of certificates returned in the header, while individual certificate data would be stored in `recordSet` elements. Variables in the header are accessed as `result.header.`*`variableName`*.

- The `fixed` object contains information that is fixed and independent of the data being returned in the response. Such data includes the hostname and port of the CMS server that handled the request, which is useful for creating HTTP forms that use the server and port that generated the response. Variables in the `fixed` object are accessed as `result.fixed.`*`variableName`*.

- The `recordSet` array is available to any response that returns certificates. Data for each certificate is returned in variables in a `record` object. Each `record` object is then added to the `recordSet` array. Variables in a particular record object are accessed by an index into the `recordSet` array: `result.recordSet[`*`i`*`].`*`variableName`*.

The following table describes the format an purpose of all the data included in the base `<CMS_TEMPLATE>` JavaScript:

Table 24.9 Variables Returned by the Base JavaScript

Variable	Format/Type and Description

`AUTH_TOKEN`	array
	Each element in this array is a name-value pair. These pairs represent variables that were returned from an authentication plug-in used (internally) by the interface. For example, if the authentication plug-in returned variables `fooValid` and `barValid` indicating whether the `foo` and `bar` parameters of a request were valid, the JavaScript in the response might look like:
	`header.AUTH_TOKEN[0].name = "fooValid";`
	`header.AUTH_TOKEN[0].value = "false";`
	`header.AUTH_TOKEN[1].name = "barValid";`
	`header.AUTH_TOKEN[1].value = "true";`
	These values are created when an authentication plug-in invokes `set(nameSrting, valueString)` on an `AuthToken` object.

`authTokenCount`	number The number of `AUTH_TOKEN` objects returned in this response.
`fixed`	object An object for containing data that is constant, such as the hostname, port number, or ID number associated with a request.
`header`	object An object for containing data that applies to the entire response, such as the number of records or the LDAP query that was used to get the data.

Table 24.9 Variables Returned by the Base JavaScript (Continued)

Variable	Format/Type and Description

HTTP_HEADERS	Array
	Each element in this array is a name-value pair. These pairs represent HTTP request headers sent from the client to the interface.
	
	Use the CMS.cfg parameter saveHttpHeaders to list HTTP header values that should be saved and returned in responses.
	
	For example, if saveHttpHeaders is set to "accept-language, user-agent", the JavaScript in the response might look like:
	
	`header.HTTP_HEADERS[0].name = "accept-language"`
	`header.HTTP_HEADERS[0].value = "en";`
	`header.HTTP_HEADERS[1].name = "user-agent"`
	`header.HTTP_HEADERS[1].value = "Mozilla/4.51 (X11; U; SunOS 5.7 sun4u)";`
	
	The default value for saveHttpHeaders (if it is not explicitly set in CMS.cfg) is "accept-language, user-agent".

httpHeadersCount	number The number of HTTP_HEADERS objects returned in this response.

HTTP_PARAMS	Array
	Each element in this array is a name-value pair. These pairs represent variables and their values that were used in the HTTP request made to the interface.
	
	Some HTTP parameters will be discarded after the request has been authenticated so that sensitive data is not stored on the CMS server. By default, parameters named pwd, password, and passwd are discarded after authentication. All other parameters are passed back in the response. Use the CMS.cfg parameter dontSaveHttpParams to list parameters that should be discarded by the CMS server after authentication.
	
	For example, if dontSaveHttpParams is set to "pwd" and the request used parameters named uid and pwd, the JavaScript in the response might look like:
	
	`header.HTTP_PARAMS[0].name = "uid"`
	`header.HTTP_PARAMS[0].value = "userOne";`
	
	The default value for dontSaveHttpParams (if it is not explicitly set in CMS.cfg) is "pwd, password, passwd".

httpParamsCount	number The number of HTTP_PARAMS objects returned in this response.

Table 24.9 Variables Returned by the Base JavaScript (Continued)

Variable	Format/Type and Description
`fixed.preserved`	any data This variable contains the value of the parameter named `preserved` passed to the interface. It allows any data to be passed through from a request to the response template by submitting a parameter named `preserved` in the request.
`record`	Object An object for containing data about a single certificate. Variables and values are added to a `record` object, then the `record` object is added as an element in the `recordSet` array.

`recordCount`	number
	The number of `record` objects returned in this response. Usually this is incremented for each `record` added to the `recordSet` array. For example,
	`recordCount = 0;`
	`record.serialNumber = 1;`
	`recordSet[recordCount++] = record;`
	`record.serialNumber = 2;`
	`recordSet[recordCount++] = record;`

`recordSet`	Array This array contains any number of `record` objects. The base JavaScript will also add `recordSet` to the `result` object, so use `result.recordSet[`*`i`*`].`*`variableName`* to access individual fields of a given `recordSet` element.
`result`	Object The primary container for all of the results returned in this template. The `fixed`, `header`, and `recordSet` objects are added to the `result` object as the last statements in the base JavaScript.

SERVER_ATTRS	Array
	Each element in this array is a name-value pair. These pairs represent variables and their values that were added by modules internal to the CMS serever or policy plug-ins. For example, if a policy plug-in added a variable called `requestStatus` with a value of `pending`, the resulting JavaScript might be:
	`header.SERVER_ATTRS[0].name = "requestStatus";`
	`header.SERVER_ATTRS[0].value = "pending";`

`serverAttrsCount`	number The number of SERVER_ATTRS objects returned in this response.

End-entity Interface Reference

This section provides a detailed reference of all the service interfaces available on an end-entity port. For each interface, there is

- a description including the URI used and the purpose
- a list of forms that use the interface by default
- a detailed description of valid input parameters and their values,
- information about the response which lists the templates used and the additional JavaScript variables available.

The following table lists the end-entity interfaces and their functions:

Table 24.10 End-Entity Interfaces

Interface	URI	Purpose
Certificate Enrollment Protocol Interface	`/pkiclient.exe`	Process Simple Certificate Enrollment Protocol (SCEP) certificate requests from routers and other VPN clients.
Challenge Revocation Interface	`/challenge_revocation1`	Revoke a certificate using a challenge phrase set during enrollment.
Display Certificate By Serial Number Interface	`/displayBySerial`	Retrieve a certificate with a given serial number in human-readable form (use Get Certificate By Serial Number to get a binary form).
Display Certificate From Request Interface	`/displayCertFromRequest`	Used on a Registration Manager to display the certificate issued for a given request identifier.
Enrollment Interface	`/enrollment`	Process manual or automated certificate requests.
Get CA Chain Interface	`/getCAChain`	Retrieve the certificate authority's certificate or certificate chain (if the CA is not self-signed).
Get Certificate By Serial Number Interface	`/getBySerial`	Get a certificate with a given serial number in a binary format (for example, PKCS #7 or a CMMF response).

Table 24.10 End-Entity Interfaces (Continued)

Interface	URI	Purpose
Get Certificate From Request Interface	`/getCertFromRequest`	Use the id assigned to a pending request to retrieve the certificate once it has been issued.
Get CRL Interface	`/getCRL`	Retrieve the Certificate Revocation List.
List Certificates Interface	`/listCerts`	List certificates based on flexible query criteria.
Renewal Interface	`/renewal`	Process requests for renewing a certificate presented to the interface using SSL client authentication.
Revocation Interface	`/revocation`	Process requests for manual revocation or for revocation of a certificate presented to the interface using SSL client authentication.

Certificate Enrollment Protocol Interface

Description

URI: `/cgi-bin/pkiclient.exe`

Available on: Certificate Manager and Registration Manager

Function: Handles Certificate Enrollment Protocol (CEP) requests from devices such as Virtual Private Network (VPN) routers.

VPN routers use CEP to enroll in and get information about their PKI. The Certificate Enrollment Protocol interface uses CEP to issue new certificates, distribute Certificate Revocation List (CRL) data, and distribute the CA certificate.

Default Forms

There are no forms that use the Certificate Enrollment Protocol. The interface is provided so that VPN clients, such as routers, can use CEP to interact with the PKI.

Request Parameters

You will not generally develop your own request forms or response templates for use with CEP. The Certificate Enrollment Protocol interface complies with the CEP protocol developed by Cisco, so if your application or device uses this protocol it will be able to use the Certificate Enrollment Protocol Interface.

To use the interface with a Cisco router, for example, you configure the router to point to the end-entity gateway port using the router's enrollment url command. You can then use `crypto ca enroll` to request a certificate:

```
> crypto ca identity Example
> enrollment url https://example:443/
> crypto ca enroll Example
```

The router uses the CEP protocol and expects to find the `/cgi-bin/pkiclient.exe` interface at the URL named by the `enrollment url` command. The details of interacting with the interface are handled by the protocol itself.

Challenge Revocation Interface

Description

URI: `/challenge_revocation1`

Available on: Certificate Manager and Registration Manager

Function: Allows an entity to revoke a certificate using a challenge password set during enrollment.

The Challenge Revocation interface is useful if an entity must revoke a certificate that is not available or not valid for SSL client authentication. (The Revocation Interface can be used to present a certificate using SSL client

authentication for revocation.) To use this interface, the challenge password for revocation must be set during enrollment (see the `challengePassword` request parameter in "Enrollment Interface" on page 936).

Default Forms

The `ChallengeRevoke1.html` form is the only default form that uses the Challenge Revocation interface. It allows an end user to enter either the certificate's serial number or subject name and the challenge password to revoke the certificate.

Request Parameters

The following table lists the parameters accepted by the Challenge Revocation interface.

Table 24.11 Parameters Accepted by the Challenge Revocation Interface

Parameter	Format and Description
`certSerialToRevoke`	number (decimal or hexadecimal) The serial number of the certificate to revoke. Either this parameter or `subjectName` are required.
`challengePhrase`	string The challenge phrase, set during certificate enrollment, that allows the certificate to be revoked.
`reasonCode`	0-8 The code for the reason why the certificate is being revoked. This code is added to the Certificate Revocation List (CRL) entry for the revoked certificate. Valid `reasonCode` values are: • `0` - Reason not specified • `1` - Key compromised • `2` - CA key compromised • `3` - Affiliation changes • `4` - Certificate superseded • `5` - Cessation of operation • `6` - Certificate is on hold

Table 24.11 Parameters Accepted by the Challenge Revocation Interface (Continued)

Parameter	Format and Description
`subjectName`	Distinguished Name (DN) string. See RFC 2253. The DN appearing in the certificate subject field. Either subjectName or certSerialToRevoke must be specified. The subject name should only be used when the subject DN is guaranteed to uniquely identify the certificate. Example DN: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`templateType`	`RevocationConfirmation` This parameter specifies which response template to use. At this time, the only valid value is `"Revocation Confirmation."` This value causes the `revocationResult.template` file to be used.

Response

The response from the Challenge Revocation interface will be identical to a response from the Revocation interface. See the Response section in "Revocation Interface" on page 968 for details on what JavaScript variables are returned in the response template.

Display Certificate By Serial Number Interface

Description

URI: `/displayBySerial`

Available on: Certificate Manager

Function: Displays a single certificate in human-readable form.

The Display Certificate By Serial Number interface is typically used within a form that lists certificates to display detailed information about a selected certificate. The response is an HTML page built from a template (not just raw certificate data), so this interface should not be used to retrieve certificates for processing (such as importing into a browser); use the Get Certificate By Serial Number Interface (`/getBySerial`) instead.

Default Forms

The Display Certificate By Serial Number interface is used in the `queryCert.template` file. Each certificate in the list of certificates satisfying the query has a button the user can press to see the certificate in detail. This button submits data to the Display Certificate By Serial Number interface.

Request Parameters

The following table lists the parameters accepted by the Display Certificate By Serial Number interface.

Table 24.12 Parameters Accepted by the Display Certificate By Serial Number Interface

Parameter	Format and Description
`op`	`displayBySerial` Specifies the operation to perform. The only valid value is `displayBySerial`.
`serialNumber`	number The serial number of the certificate to display.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `displayBySerial.template.` The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Display Certificate By Serial Number interface adds the JavaScript variables listed in the following table:

Table 24.13 Variables Returned by the Display Certificate By Serial Number Interface

Variable	Description
result.header variables	Variables added to the header object.
`authorityid`	`ca` Indicates the source of the certificate information. Only Certificate Managers can return certificates by serial number directly.
`certChainBase64`	base-64 encoded data Contains the certificate in PKCS #7 format.
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string Contains details about the certificate in a human-readable form. This is the field used to show the certificate to a user in a page.
`serialNumber`	number The serial number of the certificate in decimal.

Display Certificate From Request Interface

Description

URI: `/displayCertFromRequest`

Available on: Certificate Manager or Registration Manager

Function: Retrieves the certificate associated with an enrollment or renewal request to be displayed in a response template.

The Display Certificate From Request interface is typically used in JavaScript embedded in the response template of an enrollment or renewal request. This interface uses the `requestID` returned in the JavaScript of a response to fetch the associated certificate.

In the `requestStatus.template` file, there is JavaScript code to build a URL that fetches the certificate from the Display Certificate From Request Interface if the CMS server is a Registration Authority.

The `requestID` parameter from the response template is required: it identifies the request from which to extract the certificate.

Default Forms

By default, the Display Certificate From Request interface is used by the `requestStatus.template` file only.

Request Parameters

The following table lists the parameters accepted by the Display Certificate From Request interface.

Table 24.14 Parameters Accepted by the Display Certificate From Request Interface

Parameter	Format and Description
`requestId`	number The `requestID` returned in the JavaScript by the Enrollment or Renewal interface (`fixed.requestID`).
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

By default, the `displayCertFromRequest.template` file is used to create the response. The `<CMS_TEMPLATE>` tag is replaced with the the base JavaScript for responses. In addition, the Get Certificate From Request interface adds the JavaScript variables listed in the following table:

Table 24.15 Variables Returned by the Display Certificate From Request Interface

Variable	Description
result.fixed variables	Variables added to the `fixed` object.
`authorityName`	`Certificate Manager` | `Registration Manager` The name of the system that handled the request.
`errorDescription`	string A message providing more details about the error described in `errorDetails`. This variable is only present if an error occurred while processing the request.
`errorDetails`	string A message explaining the error that occurred while processing the enrollment request. This variable is only present if an error occurred while processing the request.
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same port.
`port`	number The port number that was used to service the request.
`requestId`	number The request identification number that was requested.
`scheme`	`http` | `https` The protocol that was used to make the request. Use this along with `host` and `port` to make sure any new requests to the end-entity port use the correct scheme.
result.header variables	Variables added to the header object.
`emailCert`	`true` | `false` If true, the certificate contained in the `recordSet` array or `cmmfResponse` is a valid S/MIME certificate.
`noCertImport`	`true` | `false` Indicates whether the certificate should not be imported.

Table 24.15 Variables Returned by the Display Certificate From Request Interface (Continued)

Variable	Description
`requestId`	number The request identification number that was requested.
result.recordSet[i] variables	Variables added to each `record` object. Each `record` object is added as an element of the `recordSet` array. Multiple records may be returned if more than one certificate was generated as a result of the request. Dual-key requests (for example, if the request parameter `requestFormat = crmf`) may return two certificates if the request is successfully processed and approved.
`base64Cert`	string The newly issued certificate in base-64 encoded format. This string includes the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string A long text string that shows all of the certificate data in a human readable form.
`serialNo`	number The serial number (in decimal) of the certificate.

Enrollment Interface

Description

URI: `/enrollment`

Available on: Certificate Manager and Registration Manager.

Function: Enrolls an entity into the Public-Key Infrastructure (PKI).

This servlet uses data from an `HTTP POST` or `HTTP GET` to formulate a certificate request, hands the request off to a Certificate Manager, and returns a response (which may include the newly issued certificate) to the entity.

The certificate request may be based only on the data in the request, or it may get additional data from an authentication plug-in named in the `authenticator` parameter. If an authentication plug-in is used, the Certificate Manager may be able to automatically issue a certificate which is passed to the entity in the enrollment servlet's response. If no authentication plug-in is used, the request is placed in an agent queue for manual approval and the enrollment servlet returns a "request pending" page to the entity.

Note The forms rely on a shared library called `xenroll.dll` (downloaded from the CMS server) to generate keys for Microsoft Internet Explorer browsers. By default, the keys generated by `xenroll.dll` have a "medium" security setting which means they will be stored unencrypted and that they can be used by the browser for signing without prompting the user for a password. A "high" security setting will store the keys in a separate, encrypted file and force the user to enter a password to use the keys for signing. There is no way to force a "high" setting for keys, but you can force a dialog to appear to allow the user to choose a security setting when the key is first generated. Edit the the VisualBasic script for `xenroll.dll` used in the enrollment forms (listed in the next section). Set the value of the `GenKeyFlags` parameter to 3 to prompt the user for a security setting when a key is generated using Microsoft Internet Explorer.

Default Forms

There are two types of default HTML forms that use the enrollment interface: manual or automated enrollment. Forms that use automated enrollment send an authentication plug-in name as a parameter in the request which the servlet can use to authenticate and process the request without manual intervention.

The default manual enrollment forms are:

- `ManUserEnroll.html` for requesting client certificates.
- `ManServerEnroll.html` for requesting server certificates.
- `ManObjSign.html` for requesting object signing certificates.
- `ManCAEnroll.html` for requesting subordinate Certificate Manager signing certificates.
- `ManRAEnroll.html` for requesting Registration Manager certificates.

The default automated enrollment forms are:

- `DirUserEnroll.html` uses a `UserDirEnrollment` instance of the `UidPwdDirAuth` plug-in class by default.
- `DirPinUserEnroll.html` uses a `PinDirEnrollment` instance of the `UidPwdPinDirAuth` plug-in class by default.

Request Parameters

The following table lists the parameters accepted by the enrollment interface.

Table 24.16 Parameters Accepted by the Enrollment Interface

Parameter	Format and Description
Subject Name	
`subject`	Distinguished Name (DN) string. See RFC 2253. DN to be used for the certificate subject. Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
Contact Information	
`csrRequestorName`	string Name of the entity making a request; helps identify the requestor during manual enrollment. Example: Alice Apple
`csrRequestorEmail`	string Email address of the entity making a request. May be used to send out notification when a certificate has been issued. Example: `alice@example.com`
`csrRequestorPhone`	string Phone number of the entity making a request. Example: `650.555.1212`
`csrRequestorComments`	string Additional comments provided by the requestor on the HTML form. This field can be used if there is additional information you want to collect to help the manual enrollment.
Netscape Certificate Type Extensions	Parameters for setting bits in the `netscape-cert-type` certificate extension. See `http://home.netscape.com/eng/security/comm4-cert-exts.html` for details. A `true` value sets the bit to 1; `false` sets the bit to 0.

Table 24.16 Parameters Accepted by the Enrollment Interface (Continued)

Parameter	Format and Description
`email`	`true` | `false` Sets the S/MIME client certificate bit (bit 2).
`email_ca`	`true` | `false` Sets the S/MIME certificate issuer bit (bit 6).
`object_signing`	`true` | `false` Sets the object signing certificate bit (bit 3).
`object_signing_ca`	`true` | `false` Sets the object signing certificate issuer bit (bit 7).
`ssl_ca`	`true` | `false` Sets the SSL certificate issuer bit (bit 5).
`ssl_client`	`true` | `false` Sets the SSL client authentication certificate bit (bit 0).
`ssl_server`	`true` | `false` Sets the SSL server authentication certificate bit (bit 1).
Key Usage	Parameters for setting bits in the `keyUsage` certificate extension. A `true` value sets the bit to 1; `false` sets the bit to 0.
`crl_sign`	`true` | `false` Sets the `keyUsage` extension bit (6) indicating that the key may be used to sign Certificate Revocation Lists (CRLs).
`data_encipherment`	`true` | `false` Sets the `keyUsage` extension bit (3) indicating that the key may be used to encipher application data (as opposed to key material).
`decipher_only`	`true` | `false` Sets the `keyUsage` extension bit (8) indicating that the key may **only** be used to decipher data and keys. If this parameter is `true`, `keyAgreement` should also be `true`.
`digital_signature`	`true` | `false` Sets the `keyUsage` extension bit (0) indicating that the key may be used to sign any data. This parameter should be `true` for SSL client certificates, S/MIME signing certificates, and object signing certificates.
`encipher_only`	`true` | `false` Sets the `keyUsage` extension bit (7) indicating that the key may **only** be used to encipher data and keys. If this parameter is `true`, keyAgreement should also be true.

Table 24.16 Parameters Accepted by the Enrollment Interface (Continued)

Parameter	Format and Description
`key_agreement`	`true` | `false` Sets the `keyUsage` extension bit (4) indicating that the key may be used to encipher and decipher keys during key agreement.
`key_certsign`	`true` | `false` Sets the `keyUsage` extension bit (5) indicating that the key may be used to sign other certificates. All CA signing certificates should set this parameter to `true`.
`key_encipherment`	`true` | `false` Sets the `keyUsage` extension bit (2) indicating that the key may be used to encipher symmetric session keys. This parameter should be `true` for SSL server and S/MIME encryption certificates.
`non_repudiation`	`true` | `false` Sets the `keyUsage` extension bit (1) indicating that the key may be used to create non-repudiable (by the signer) digital signatures. Non-repudiation service requires more infrastructure, planning, and policy than just setting this bit. Consider the ramifications before using this bit
Automated Enrollment	Parameters to configure automatic authentication for entity requests.
`authenticator`	string Specifies the name of the authentication plug-in instance to use to authenticate the entity.
`uid`	string Specifies a unique identifier passed to the authentication plug-in.
`pin`	string An optional identifying string that helps to authenticate an entity. Usually used when the Pin Generator tool has been used to populate a directory with unique identifiers for each user.
`pwd`	string Specifies the password passed to the authentication plug-in.
Other	
`certNickname`	string Specifies the nickname that should be associated with the certificate in the reply; used with Certificate Request Management Format (CRMF) requests.

Table 24.16 Parameters Accepted by the Enrollment Interface (Continued)

Parameter	Format and Description
certType	ca | CEP-Request | client | objSignClient | ra | server | other Specifies the type of certificate requested by the entity. The default is client. The certType is not associated with any certificate extensions. It may be used by policy modules to make decisions, and it may be used by a CMS server to determine how to decode the request or format the response.
challengePassword	string An optional challenge phrase or password that can be used later by the entity to revoke the certificate. This parameter is optional. If you use this, entities can use the "Challenge Revocation Interface" (/challenge_revocation1, page 929)with this challenge password to revoke a certificate without manual intervention and without SSL client authentication.
CRMFRequest	base-64 encoded data If requestFormat = crmf, this parameter should be used to send the base-64 encoded CRMF request.
importCAChain	true | false Used only when importCert = true. The default, if this parameter is not explicitly passed, is true. If set to true, a successful certificate request will return a PKCS #7 formatted certificate chain; if set to false, a single, DER-encoded certificate will be returned. The certificate chain includes the issued certificate and the CA (issuer) certificate.
importCert	true | false If true, and the certificate request is not deferred or rejected, the CMS server's response will be binary data with the MIME type determined by the importCertMimeType parameter. The data returned will be either a certificate or a certificate chain, based on the value of importCAChain.
importCertMimeType	MIME Type string Sets the MIME type the CMS server uses when a certificate is returned to the requestor. The default is application/x-x509-user-cert. The MIME type should be in the standard MIME type format of <type>/<subtype>.
pkcs10Request	base-64 encoded data If requestFormat = pkcs10, this parameter should be used to send the base-64 encoded certificate request.

Table 24.16 Parameters Accepted by the Enrollment Interface (Continued)

Parameter	Format and Description
`requestFormat`	`clientAuth` | `crmf` | `keygen` | `pkcs10` The value indicates the format used to submit the certificate request: • `clientAuth` - information for the new request is taken from the certificate presented by the client during SSL client authentication. • `crmf` - the certificate request is a base-64 encoded blob contained in the `CRMFRequest` parameter. • `keygen` - the certificate request is a base-64 encoded blob generated using the HTML `<KEYGEN>` tag. It is contained in the `subjectKeyGenInfo` parameter. • `pkcs10` - the certificate request is a base-64 encoded blob contained in the `pkcs10Request` parameter.
`subjectKeyGenInfo`	base-64 encoded data If `requestFormat=keygen`, this parameter should be used to send the base-64 encoded `keygen` request. To use the `<KEYGEN>` HTML tag to cause the browser to generate the request using this parameter, the format is `<KEYGEN name="subjectKeyGenInfo">`
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The Registration Authority or Certificate Authority that process an enrollment request will perform some processing, determine the status of the request, then return a result using the appropriate template for the status.

The response templates are ASCII files that you can edit to create responses suited to your needs. The templates may include JavaScript that depends on the JavaScript inserted in place of the `<CMS_TEMPLATE>` tag when the response is sent. The status of the request determines which template will be used for the

response. The following table describes the templates used by the enrollment interface (more details on the request status codes can be found in Table 24.18):

Table 24.17 Enrollment Interface Response Templates

Template File Name	Request Status	Description
`EnrollSuccess.template`	2 (Success)	Used only for requests that specify an authenticator. If authentication and subsequent policy processing are successful and `importCert` was "`true`" in the request, a certificate is generated (otherwise, see `GenRejected.template`). The issued certificate is included in base-64 in the response. The template includes JavaScript and VisualBasic code that attempts to import the certificate into a browser's certificate database.
`GenError.template`	6 (Error)	Used to display an error message.
`GenPending.template`	3 (Pending)	Used to inform the user that the certificate request he or she submitted has been queued for agent approval.
`GenRejected.template`	5 (Rejected)	Used to inform the user that the certificate request he or she submitted has been rejected by the CMS server.
`GenSuccess.template`	2 (Success)	Used to inform the user that the certificate request he or she submitted has been approved by the CMS server.
`GenSvcPending.template`	3 (Pending)	Used to inform the user that the certificate request he or she submitted has been queued for agent approval.
`GenUnauthorized.template`	1 (Unauthorized)	Used to inform the user that he or she performed an unauthorized operation.

The `<CMS_TEMPLATE>` tag in the selected template is replaced with the base JavaScript code. In addition, the Enrollment interface may add the JavaScript variables listed in the following table. Not all templates use all of the variables

listed in the table; a variable is only included when it is used (for example, the `EnrollmentSuccess.template` does not include `result.fixed.unexpectedError`).

Table 24.18 Variables Returned by the Enrollment Interface

Variable	Description
result.fixed variables	Variables added to the `fixed` object.
`authorityName`	`Certificate Manager` | `Registration Manager` The name of the system that handled the request.
`certType`	`ca` | `CEP-Request` | `client` | `objSignClient` | `ra` | `server` | `other` The type of certificate returned. This value is the same as the `certType` value passed to the interface in the request.
`errorDescription`	string A message providing more details about the error described in `errorDetails`. This variable is only present if an error occurred while processing the request.
`errorDetails`	string A message explaining the error that occurred while processing the enrollment request. This variable is only present if an error occurred while processing the request.
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same port.
`port`	number The port number that was used to service the request.
`requestId`	number A unique number assigned by the CMS server to this request. This is especially useful for pending requests since there is no unique certificate serial number yet assigned.

Table 24.18 Variables Returned by the Enrollment Interface (Continued)

Variable	Description
`requestStatus`	number A code indicating the current status of the request: • 1 (Unauthorized): The request specified a value for an authenticator to perform an automated enrollment, and the authenticator did not authorize the request. • `2` (Success): Processing the request was successful and a certificate has been issued. If `importCert` was set to `true`, the response will include code (from the `EnrollSuccess.template`) to import the certificate into the browser making the request. Otherwise, the response is only a success message. • `3` (Pending): The request has been successfully processed by the CMS server and added to a queue for approval by an agent. If the request has been submitted to another Certificate Manager or Data Recovery Manager and is currently pending in the queue for that service, the response template will be `GenSvcPending.template` instead of `GenPending.template`. • `4` (Reserved): Not currently used. • `5` (Rejected): The request was rejected during policy processing. • `6` (Error): An error occurred when the CMS server processed the request. The error may be the result of missing or improperly formatted parameters. • `7` (Exception): An unknown or unexpected error occurred when the CMS server processed the request.
`scheme`	`http` | `https` The protocol that was used to make the request. Use this along with `host` and `port` to make sure any new requests to the end-entity port use the correct scheme.
`unexpectedError`	string A message explaining the exception or unexpected error that occurred.
result.recordSet[i] variables	Variables added to each `record` object. Each `record` object is added as an element of the `recordSet` array. Multiple records may be returned if more than one certificate was generated as a result of the request. Dual-key requests (for example, if the request parameter `requestFormat = crmf`) may return two certificates if the request is successfully processed and approved.
`base64Cert`	string The newly issued certificate in base-64 encoded format. This string includes the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.

Table 24.18 Variables Returned by the Enrollment Interface (Continued)

Variable	Description
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string A long text string that shows all of the certificate data in a human readable form.
`policyMessage`	string If the request was rejected by policy processing on the CMS server, this variable will contain a message explaining why.
`serialNo`	number The serial number (in decimal) of the newly issued certificate.

Get CA Chain Interface

Description

URI: `/getCAChain`

Available on: Certificate Manager only.

Function: Retrieves the CA certificate or certificate chain for the Certificate Manager either in binary form for use by an application or in a format for display.

The Get CA Chain interface accepts an operation (for example, download) and a MIME type to be used for the response. The response is always the CA certificate or certificate chain (if the CA certificate is not self-signed) for the Certificate Manager handling the request. No templates are used; the response is either ASCII data that can be displayed or a binary blob (using the indicated MIME type) that can be used by the requesting application.

Using the Get CA Chain interface to display certificates is useful for creating data that can be imported into another application such as an HTTP or LDAP server.

Default Forms

The Get CA Chain interface uses one default form: `GetCAChain.html`. This form allows an entity to choose one of four operations:

- Import the CA certificate chain into a browser (`download`).
- Download the CA certificate chain in binary form (`downloadBIN`).
- Display the CA certificate chain in PKCS #7 for importing into a server (`display`).
- Display certificates in the CA certificate chain for importing individually into a server (`displayIND`).

Request Parameters

The following table lists the parameters accepted by the Get CA Chain interface.

Table 24.19 Parameters Accepted by the Get CA Chain Interface

Parameter	Format and Description
`mimeType`	`<type>/<subtype>` Indicates the MIME the CMS server should use for the response. The default form uses `application/x-x509-ca-cert` for downloads. For `op = downloadBIN`, `mimeType` is ignored and `application/octet-stream` is always used.

Table 24.19 Parameters Accepted by the Get CA Chain Interface (Continued)

Parameter	Format and Description
`op`	`display` | `displayIND` | `download` | `downloadBIN` This required parameter specifies how the CA certificate chain should be returned: • `display` returns a base-64 encoded PKCS #7 certificate chain. • `displayIND` returns each certificate in the CA chain in a base-64 encoded DER blob and a human-readable format. • `download` returns the entire certificate chain in binary form using the MIME type specified. If the browser is not Internet Explorer, the chain is returned as a PKCS #7 blob. If the browser is Internet Explorer, only the CA signing certificate will be returned in a DER-encoded format. • `downloadBIN` is the same as download, except that the MIME type is always set to `application/octet-stream`. Use of `downloadBIN` is deprecated; use `download` and set `mimeType = application/octet-stream` in the request instead.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The Get CA Chain interface does not use any templates. The response is just the requested certificate or certificates in the format indicated by the request parameters.

Get Certificate By Serial Number Interface

Description

URI: `/getBySerial`

Available on: Certificate Manager only

Function: Retrieves the certificate with the given serial number in a specified format. The certificate can be imported into a browser.

This interface is used in the `EnrollSuccess.template` and `RenewalSuccess.template` to download and import the newly issued certificate. The `displayBySerial.template` also uses this interface to create "Import Certificate" and "Import S/MIME Certificate" buttons on a page that displays a certificate; this allows a user to retrieve and import a certificate that was issued manually.

Default Forms

There are no default forms that use the Get Certificate By Serial Number interface. This interface is usually used embedded in a response template to either embed a certificate in the response or provide a button on a form that downloads and imports the certificate.

Request Parameters

The following table lists the parameters accepted by the Get Certificate By Serial Number interface.

Table 24.20 Parameters Accepted by the Get Certificate By Serial Number Interface

Parameter	Format and Description
`cmmfResponse`	`true` | `false` Indicates whether the certificate should be returned using CMMF. The CMMF format can be used with browsers that understand it to allow the browser to import the certificate into its database. The CMMF data will be returned as a JavaScript variable in the response.
`emailCert`	`true` | `false` Indicates whether the certificate returned is expected to be valid for signing e-mail.
`importCert`	`true` | `false` Indicates how to format the certificate in the response if `cmmfResponse` is `false` or not set. By default, the response will be a page created from the ImportCert.template file. If `importCert = true`, the MIME type will be set to `application/x-x509-user-cert` and the data will be a binary blob in PKCS #7 format.

Table 24.20 Parameters Accepted by the Get Certificate By Serial Number Interface (Continued)

Parameter	Format and Description
`serialNumber`	number The serial number of the certificate to retrieve.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

If `importCert = true` in the request, the response is a binary blob containing the certificate and no templates are used to construct the response. See the Request Parameters for details on how the response is formatted.

By default, the `ImportCert.template` file is used to create the response. The `<CMS_TEMPLATE>` tag is replaced with the base JavaScript for responses. In addition, the Get Certificate By Serial Number interface adds the JavaScript variables listed in the following table:

Table 24.21 Variables Returned by the Get Certificate By Serial Number Interface

Variable	Description
result.fixed variables	Variables added to the `fixed` object.
`authorityName`	`Certificate Manager` | `Registration Manager` The name of the system that handled the request.
`certNickname`	string The nickname for the certificate. By default, this is just the certificate subject DN.
`certType`	`ca` | `CEP-Request` | `client` | `objSignClient` | `ra` | `server` | `other` The type of certificate returned. This value is the same as the `certType` value passed to the interface in the request.

Table 24.21 Variables Returned by the Get Certificate By Serial Number Interface (Continued)

Variable	Description
`cmmfResponse`	base-64 encoded data The CMMF response data containing the certificate (if `cmmfResponse` was `true` in the request). If the browser supports the Personal Security Manager crypto API, you can use this response with a call to `importUserCertificates` to import the certificate into a local database: `importUserCertificates(result.fixed.nickname, result.fixed.cmmfResponse, true);` (The last parameter indicates whether the user should be prompted to back up the key.)
`errorDescription`	string A message providing more details about the error described in `errorDetails`. This variable is only present if an error occurred while processing the request.
`errorDetails`	string A message explaining the error that occurred while processing the enrollment request. This variable is only present if an error occurred while processing the request.
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same port.
`port`	number The port number that was used to service the request.
`scheme`	`http` | `https` The protocol that was used to make the request. Use this along with host and port to make sure any new requests to the end-entity port use the correct scheme.
	Variables added to each `record` object. Each `record` object is added as an element of the `recordSet` array. Multiple records may be returned if more than one certificate was generated as a result of the request. Dual-key requests (for example, if the request parameter `requestFormat = crmf`) may return two certificates if the request is successfully processed and approved.
`base64Cert`	string The newly issued certificate in base-64 encoded format. This string includes the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.

Table 24.21 Variables Returned by the Get Certificate By Serial Number Interface (Continued)

Variable	Description
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string A long text string that shows all of the certificate data in a human readable form.
`serialNo`	number The serial number (in decimal) of the certificate.

Get Certificate From Request Interface

Description

URI: `/getCertFromRequest`

Available on: Certificate Manager or Registration Manager

Function: Retrieves the certificate associated with an enrollment or renewal request to be displayed in a response template or imported into a browser.

The Get Certificate From Request interface is typically used in JavaScript embedded in the response template of an enrollment or renewal request. This interface uses the `requestID` returned in the JavaScript of a response to fetch the associated certificate.

In the `EnrollSuccess.template` and `RenewalSuccess.template` files, there is JavaScript code to build a URL that fetches the certificate from the Get Certificate From Request interface. The URL is assigned to the `window.location` object, which causes the contents of the URL to be displayed in-line in the response.

The `requestID` parameter from the response template is required: it identifies the request from which to extract the certificate. A parameter can also be used to instruct the requesting browser to import the certificate into its database: `importCert` or `cmmfResponse` (for browsers that support CMMF).

Default Forms

The Get Certificate From Request interface is used by response templates, not forms that an entity submits. The interface is used in these templates to incorporate the certificate in the page itself, so that it can be displayed and possibly imported into the browser. The templates that use the interface by default are:

- `displayCertFromRequest.template` is a generic template that shows how to display a certificate from a request.
- `EnrollSuccess.template` is the template returned by a successful enrollment request (when a certificate has been issued, not when the request is successful but still pending).
- `RenewalSuccess.template` is the template returned by a successful automated renewal request (such as a renewal using SSL client authentication).

Request Parameters

The following table lists the parameters accepted by the Get Certificate From Request interface.

Table 24.22 Parameters Accepted by the Get Certificate From Request Interface

Parameter	Format and Description
`cmmfResponse`	`true` | `false` Indicates whether the returned certificate should be formatted using Certificate Management Message Format (CMMF). Entities that support CMMF can use the result to import the certificate into a local database. The CMMF data will be returned as a JavaScript variable in the response.
`importCert`	`true` | `false` Indicates whether the returned certificate should be imported into the local database. This causes the MIME type of the returned HTTP to be `application/x-x509-user-cert` and the certificate to be a PKCS #7 formatted binary blob.

Table 24.22 Parameters Accepted by the Get Certificate From Request Interface (Continued)

Parameter	Format and Description
`requestId`	number The `requestID` returned in the JavaScript by the Enrollment or Renewal interface (`fixed.requestID`).
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

If `importCert = true` in the request, the response is a binary blob containing the certificate and no templates are used to construct the response. See the Request Parameters for details on how the response is formatted.

By default, the `ImportCert.template` file is used to create the response. The `<CMS_TEMPLATE>` tag is replaced with the base JavaScript for responses. In addition, the Get Certificate From Request interface adds the JavaScript variables listed in the following table:

Table 24.23 Variables Returned by the Get Certificate From Request Interface

Variable	Description
result.fixed variables	Variables added to the `fixed` object.
`authorityName`	`Certificate Manager` | `Registration Manager` The name of the system that handled the request.
`certNickname`	string The nickname for the certificate. By default, this is just the certificate subject DN.
`cmmfResponse`	base-64 encoded data The CMMF response data containing the certificate (if `cmmfResponse` was `true` in the request). If the browser supports the Personal Security Manager crypto API, you can use this response with a call to `importUserCertificates` to import the certificate into a local database: `importUserCertificates(result.fixed.nickname, result.fixed.cmmfResponse, true);` (The last parameter indicates whether the user should be prompted to back up the key.)

Table 24.23 Variables Returned by the Get Certificate From Request Interface (Continued)

Variable	Description
`errorDescription`	string A message providing more details about the error described in `errorDetails`. This variable is only present if an error occurred while processing the request.
`errorDetails`	string A message explaining the error that occurred while processing the enrollment request. This variable is only present if an error occurred while processing the request.
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same port.
`port`	number The port number that was used to service the request.
`requestId`	number The request identification number that was requested.
`scheme`	`http` | `https` The protocol that was used to make the request. Use this along with host and port to make sure any new requests to the end-entity port use the correct scheme.
result.header variables	Variables added to the header object.
`emailCert`	`true` | `false` If true, the certificate contained in the `recordSet` array or `cmmfResponse` is a valid S/MIME certificate.
`noCertImport`	`true` | `false` Indicates whether the certificate should not be imported.
`requestId`	number The request identification number that was requested.
result.recordSet[i] variables	Variables added to each `record` object. Each `record` object is added as an element of the `recordSet` array. Multiple records may be returned if more than one certificate was generated as a result of the request. Dual-key requests (for example, if the request parameter `requestFormat = crmf`) may return two certificates if the request is successfully processed and approved.

Table 24.23 Variables Returned by the Get Certificate From Request Interface (Continued)

Variable	Description
`base64Cert`	string The newly issued certificate in base-64 encoded format. This string includes the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string A long text string that shows all of the certificate data in a human readable form.
`serialNo`	number The serial number (in decimal) of the certificate.

Get CRL Interface

Description

URI: `/getCRL`

Available on: Certificate Manager only

Function: Retrieves the current Certificate Revocation List (CRL) for this certificate authority.

This interface can be used to retrieve a CRL for display or importing into an application and it can be used simply to check whether a certificate appears on the current CRL.

Default Forms

The only default form that uses the Get CRL interface is `DisplayCRL.html`. This form allows the user to choose any of the possible options for the getCRL interface:

- Check whether a particular certificate is on the CRL.

- Import the CRL into a browser.
- Display the CRL in binary form.
- Display the CRL in human-readable form (pretty print).

Request Parameters

The following table lists the parameters accepted by the Get CRL interface.

Table 24.24 Parameters Accepted by the Get CRL Interface

Parameter	Format and Description
`certSerialNumber`	Number string in decimal (e.g., `330`) or hexadecimal (`0x14A`). If `op = checkCRL`, use this parameter to specify the serial number of the certificate to check.
`issuepoint`	`MasterCRL` The default value, `MasterCRL`, indicates that the complete master CRL should be checked. Other issue points may be configured for a Certificate Manager; in that case use the token that defines the issue point in the configuration file (`CMS.cfg`).
`op`	`checkCRL` | `displayCRL` | `getCRL` | `importCRL` This required parameter specifies the CRL operation to perform: • `checkCRL` instructs the Certificate Manager to look for the serial number specified in `certSerialNumber` on the CRL. The result is returned in the `result.header.isOnCRL` field. • `displayCRL` returns the entire CRL formatted in HTML for display in a browser. • `getCRL` returns the entire CRL as a PKCS #7 formatted blob; the MIME type of the response will be `application/octet-stream` for Communicator clients or `application/x-pkcs7-crl`. for Internet Explorer. • `importCRL` is the same as `getCRL` except the MIME type is always `application/x-pkcs7-crl.`
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

Responses to requests for `checkCRL` or `displayCRL` use the displayCRL.template template. The `<CMS_TEMPLATE>` tag is replaced with the base JavaScript for responses. In addition, the Get CRL interface adds the JavaScript variables listed in the following table:

Table 24.25 Variables Returned by the Get CRL Interface

Variable	Description
result.header variables	Variables added to the header object.
`certSerialNumber`	number If `toDo = checkCRL`, this field contains the serial number of the certificate that was checked (from `certSerialNumber` in the request).
`crlBase64`	base-64 encoded data The base-64 encoded CRL data in PKCS #7 format.
`crlPrettyPrint`	string Contains the CRL formatted for human-readable display if `op=displayCRL` in the request.
`isOnCRL`	`true` | `false` If `toDo = checkCRL`, this field indicates whether the named certificate serial number appears on the CRL.
`toDo`	`displayCRL` | `checkCRL` Indicates the type of result being returned.

List Certificates Interface

Description

URI: `/listCerts`

Available on: Certificate Manager

Function: Retrieves a list of certificates that match a query filter.

The query criteria are search filters that the interface uses to select certificates in the Certificate Manager's repository. The `queryCert.html` default form contains JavaScript code that can construct all possible filters. You should study this file for examples before you write code to construct your own forms. Valid query filter constructions are explained in the Request Parameters section.

The response is constructed using the `queryCert.template`. The listing that this template provides by default has code for displaying a listed certificate in more detail, revoking a listed certificate, or revoking all certificates listed.

Default Forms

The List Certificates interface uses two default forms:

- `queryBySerial.html` is a simple form that accepts a lower and upper bound for the range of serial numbers and the option to skip revoked or invalid certificates. This form constructs a simple query filter to select certificates that meet the user's preferences for certificate status (valid, invalid, and revoked).
- `queryCert.html` is a complex form that allows the user to specify all possible query criteria. This form makes extensive use of JavaScript to formulate a query filter for any criteria the user chooses.

Request Parameters

The following table lists the parameters accepted by the List Certificates interface.

The queryCertFilter parameter must be a valid query filter. The syntax and valid query parameters are too complex to describe in the parameter table. Details about valid parameters and values for query filters are in a separate table following the parameters.

Table 24.26 Parameters Accepted by the List Certificates Interface

Parameter	Format and Description
maxCount	number Specifies the maximum number of certificates to display on each page returned. If more than maxCount certificates match the search criteria, each page will have controls to see the next or previous page of results.
op	listCerts The only operation supported by the List Certificates interface is listCerts.
queryCertFilter	([<OP>]<FILTER>[<FILTER>...]) Details about building query filters are provided in the next table. The queryCertFilter must be enclosed in parentheses. The <OP> argument, required if there is more than one <FILTER>, specifies how the filters that follow should be logically evaluated: • & (ampersand) means that the filters should be linked by a logical AND: all filters must evaluate to true for the expression to match a certificate. • | (pipe) means that the filters should be linked by a logical OR: if at least one filter evaluates to true, the expression matches a certificate. Any number of filters can be concatenated within any set of parentheses. An example filter is (&(certStatus=VALID)(|(x509cert.nsExtension.SSLClient=on)(x509cert.nsExtension.SecureEmail=on))) This filter matches any certificate that is valid and has either the SSL Client or S/MIME bit set in the netscape-cert-type extension.
querySentinel	number number The querySentinel indicates which record out of the total matching set should be the first displayed on the resulting output page. For example, if totalRecordCount = 15 and maxCount = 5, set querySentinel to 6 to show the 6th through 10th element of the matching set.
templateName	string Filename relative to the template directory (web/ee, web/agent/ca, web/agent/kra, or web/agent/ra) of a file to use as the response template. This template will be used for any response, overriding default template settings.

The following table describes the parameter names that are valid for constructing query filters, and the range of valid values that can be used with the parameter. The parameters can be combined using parentheses and logical operators (as described in the previous table) to construct query filters of arbitrary complexity.

In a filter, the parameter name is compared to the expression value using one of the relational operators = (matches), < (less than), <= (less than or equal to), > (greater than), or >= (greater than or equal to). Some expressions (such as `x509cert.subject`) accept the asterisk (*) as a wildcard to match 0 or more characters; for example. `"E=jdoe*"` matches `"E=jdoe@netscape.com"` and `"E=jdoe@example.com."`

Table 24.27 List Certificates queryCertFilter Parameters

Parameter	Expression Values

`certCreateTime`	Value: date (number of seconds since Jan 1, 1970)
	A date object can be created using the JavaScript `Date()` constructor.
	This parameter matches the date a certificate was issued. For example, to find certificates created during 1999:
	`Object lowDate = new Date(1999,00,01);`
	`Object highDate = new Date(1999,11,31);`
	`form.queryCertFilter.value =`
	`"(&(certCreateTime>=" + lowDate + ")" +`
	`"(certCreateTime<=" + highDate + ")";`

`certIssuedBy`	Value: user ID of an agent issuing a certificate Use the asterisk (*) wildcard to match partial names. For example, `(certIssuedBy=localAgent*)`.
`certRecordId`	Value: number in decimal or hexadecimal. This parameter matches the serial number on a certificate. Connect a lower and upper bound with a logical AND (&) to specify a bounded range of serial numbers. For example: `(&(certRecordId>=100)(certRecordId<=199))`
`certRevokedBy`	Value: user ID of an agent that revoked a certificate Use the asterisk (*) wildcard to match partial names. For example, `(certRevokedBy=localAgent*)`.
`certRevokedOn`	Value: date (number of seconds since Jan 1, 1970) A date object can be created using the JavaScript `Date()` constructor. This parameter matches the date when a certificate was revoked. See `certCreateTime` for an example of creating a date value in JavaScript

Table 24.27 List Certificates queryCertFilter Parameters (Continued)

Parameter	Expression Values
`certStatus`	Value: `*` | `EXPIRED` | `INVALID` | `REVOKED` | `VALID` | `REVOKED_EXPIRED` This parameter matches the current status of a certificate. The asterisk (*) matches any status.
`x509cert.certRevoInfo`	Value: `*` | number between 0 and 6 This parameter matches the reason for revocation code on a certificate. The revocation codes are: • `0` - Reason not specified • `1` - Key compromised • `2` - CA key compromised • `3` - Affiliation changes • `4` - Certificate superseded • `5` - Cessation of operation • `6` - Certificate is on hold To search for multiple values, construct a filter with multiple `x509cert.certRevoInfo` parameters connected with a logical OR. Do not connect these parameters with an AND, since a certificate cannot have more than one revocation reason. For example, to match certificates revoked due to key compromise or an unspecified reason: `(|(x509cert.certRevoInfo=0)(x509cert.certRevoInfo=1))`
`x509cert.duration`	Value: number of milliseconds This parameter matches the total number of milliseconds of a certificates validity period. Typically a range is specified using filters with `>=` and `<=` operators, rather than an exact match. The following list shows the number of milliseconds in some typical time intervals: • Day: 86,400,000 • Week: 604,800,000 • Month (30 Days): 2,592,000,000 • Year: 31,536,000,000
`x509cert.notAfter`	Value: date (number of seconds since Jan 1, 1970) A date object can be created using the JavaScript `Date()` constructor. This parameter matches the date when a certificate expires. See `certCreateTime` for an example of creating a date value in JavaScript

Table 24.27 List Certificates queryCertFilter Parameters (Continued)

Parameter	Expression Values
`x509cert.notBefore`	Value: date (number of seconds since Jan 1, 1970) A date object can be created using the JavaScript `Date()` constructor. This parameter matches the date when a certificate became valid. See `certCreateTime` for an example of creating a date value in JavaScript
`x509cert.nsExtension.<X>`	Value: `on` | `off` This parameter matches the bit in the ns-cert-type extension specified by the extension `<X>`. Substitute an extension identifier for `<X>`: • `SecureEmail` - for S/MIME certificates • `SSLClient` - for SSL client certificates • `SSLServer` - for SSL server certificates • `SubordinateEmailCA` - for certificates with the S/MIME CA bit set • `SubordinateSSLCA` - for certificates with the SSL CA bit set For example, to match only certificates with the SSL client bit (bit 0) set in the ns-cert-type extension: `(x509cert.nsExtension.SSLClient=on)`

`x509cert.subject`	Value: a pattern that may include the wildcard (`*`)
	This parameter matches the certificate subject DN. You can use a single filter or connect multiple filters to build more complex DN patterns.
	The value is typically a string in the form `*<TAG>=<VALUE>*`. The asterisks allow the name-value pair to be matched at any location in the DN. The tag is one of the subject DN attributes: `CN`, `E`, `UID`, `OU`, `O`, `L`, `ST`, `C`.
	To allow partial matches, use the wildcard in the attribute value. For example, to match email addresses containing `"jdoe,"`
	`(x509cert.subject=*E=*jdoe*)`
	To force an exact match of `"jdoe@example.com,"` you should still use the wildcard to allow the `E` attribute to occur anywhere in the DN:
	`(|(x509cert.subject=*E=jdoe@example.com,*)`
	`(x509cert.subject=*E=jdoe@example.com))`

Response

The default response template is `queryCert.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Revocation interface adds the JavaScript variables listed in the following table:

Table 24.28 Variables Returned by the List Certificates Interface

Variable	Description
result.header Variables	Variables added to the `header` object.
`currentRecordCount`	The total number of certificates displayed on this page of output. This number may be less than totalRecordCount.
`issuerName`	The distinguished name (DN) of the certificate authority that processed the query. This DN appears in the issuer field of all of the certificates listed. Example: CN=Certificate Manager, O=Organization, C=US
`maxCount`	The maximum number of certificate records to display on any single page of output.
`op`	The operation parameter to send (to the `serviceURL`) when the user requests more certificates. This value will always be `listCerts` for the List Certificates interface.
`queryCertFilter`	The `queryCertFilter` parameter that was used to generate the current list of certificates, and will be used for subsequent pages if the user requests to see more certificates. For information on how the filter is constructed, see the Request Parameters section. An example `queryCertFilter` is `(&(certStatus=VALID)(certRecordId>=100))`
`querySentinel`	This field holds the number of the lowest certificate serial number to display on the current page of output.
`serviceURL`	The URI to use to post requests for more certificates matching the query criteria. This variable will always be `/listCerts` for the List Certificates interface.
`templateName`	The name of the response template the CMS server should use to construct a response from the `serviceURL`. By default, this field will always be `queryCert.template` for the List Certificates interface.
`totalRecordCount`	The total number of records that match the query criteria. This number may be larger than the currentRecordCount if not all of the matching certificates can be displayed on one page of output.

Table 24.28 Variables Returned by the List Certificates Interface (Continued)

Variable	Description
recordSet Variables	These fields are added to each `record` object in the `recordSet` array. They are accessed as fields of a `recordSet` element in the JavaScript; for example, `result.recordSet[1].error`.
issuedBy	The user ID of the agent that issued the certificate.
`issuedOn`	The date when the certificate was issued. Dates are represented as number of seconds since January 1, 1970. The default template provides a function, `renderDateFromSecs`, that converts these dates to a human-readable string.
`revocationReason`	If the certificate has been revoked, this field contains the code for the reason. The revocation codes are: • 0 - Reason not specified • 1 - Key compromised • 2 - CA key compromised • 3 - Affiliation changes • 4 - Certificate superseded • 5 - Cessation of operation • 6 - Certificate is on hold
`revokedBy`	The user ID of the agent who revoked the certificate.
`revokedOn`	The date when the certificate was revoked. See the description for issuedOn for details on date values.
`serialNumber`	The serial number of the certificate (in decimal).
`signatureAlgorithm`	The Object Identifier (OID) of the algorithm used to sign the certificate. For example, `"1.2.840.113549.1.1.4"` is the OID for an MD5 with RSA signature.
`subject`	The subject distinguished name of the certificate. For example, `"CN=Jane Doe, UID=jdoe, OU=Users, O=Organization, ST=California, C=US."`
`subjectPublicKeyAlgorithm`	The OID of the key algorithm used by the public key contained in the certificate. For example, `"1.2.840.113549.1.1.1"` represents an RSA key.
`subjectPublicKeyLength`	The number of bits of the public key contained in the certificate.

Table 24.28 Variables Returned by the List Certificates Interface (Continued)

Variable	Description
`validNotAfter`	The date when the certificate expires. See the description for `issuedOn` for details on date values.
`validNotBefore`	The date when the certificate became valid. See the description for `issuedOn` for details on date values.

Renewal Interface

Description

URI: `/renewal`

Available on: Certificate Manager or Registration Manager

Function: Processes requests for certificate renewal.

The Renewal interface allows an end entity to present a certificate and have it renewed. Only certificates that can be used for SSL client authentication can be renewed using the Renewal interface.

Default Forms

The only default form used by the Renewal interface is `UserRenewal.html`. This form allows a user to renew a certificate using SSL client authentication.

Request Parameters

The following table lists the parameters accepted by the Renewal interface.

Table 24.29 Parameters Accepted by the Renewal Interface

Parameter	Format and Description
`certType`	`client` Only client certificate renewals are supported through the Renewal interface.
`doSslAuth`	`on` | `off` Set to on to force the CMS server to request an SSL client authentication certificate. Since this is the only renewal method supported, `doSslAuth` should always be set to `on`.
`requestFormat`	`clientAuth` Only client certificate renewals are supported through the Renewal interface.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

Responses from the Renewal interface are functionally equivalent to the Enrollment interface. Once the request has been submitted, the role of the Certificate Manager or Registration Manager is the same as if the request were for enrollment: the CMS server either rejects the request, queues it for manual processing, or issues a certificate.

The only difference in the response is for a successful request. The Renewal interface uses the `RenwalSuccess.template` file by default instead of `EnrollSuccess.template`. The difference between these two files (by default) is superficial: the word "Enrollment" is replaced with the word "Renewal." If you want to customize the renewal success message, customize the RenewalSuccess.template file.

Except for template for a successful request, the Renewal interface response is identical to the Enrollment interface response.

Refer to the Response section of the section "Enrollment Interface" on page 936 for complete details on the data returned and templates used.

Revocation Interface

Description

URI: `/revocation`

Available on: Certificate Manager or Registration Manager

Function: Allows automatic revocation of certificates by client authentication (an entity can revoke a certificate it presents).

The response is always a form that indicates the status of the revocation request (revoked, pending, or error), the result of updating the Certificate Revocation List, and the result of updating the certificate directory (if publishing is enabled).

Default Forms

The Revocation interface uses the UserRevocation.html form by default. This form posts requests that use SSL client authentication to present the certificate to be revoked. The certificate is automatically revoked.

Request Parameters

The following table lists the parameters accepted by the Revocation interface.

Table 24.30 Parameters Accepted by the Revocation Interface

Parameter	Format and Description
`certType`	`client` Specifies the type of certificate to revoke. For automatic revocation, the `certType` must be `client` and `doSslAuth` must be `on`.
`csrRequestorComments`	string Additional comments to assist the agent who will process the revocation request.
`csrRequestorEmail`	string Contact email address of someone responsible for the CMS server certificate. May be used to send out notification when a certificate has been revoked. Example: alice@example.com

Table 24.30 Parameters Accepted by the Revocation Interface (Continued)

Parameter	Format and Description
`csrRequestorName`	string Name of someone responsible for the CMS server certificate; helps identify the requestor during manual revocation. Example: Alice Apple
`csrRequestorPhone`	string Phone number of someone responsible for the CMS server certificate. Example: 650.555.1212
`doSslAuth`	`on | off` Instructs the CMS server to request SSL client authentication. The certificate that the entity then presents will be the one that is automatically revoked. Only valid if `certType = client`.
`op`	`RevocationRequest` RevocationRequest is the only value currently supported for the `op` parameter. This parameter is required.
`reasonCode`	`0-8` The `reasonCode` identifies the reason the certificate is being revoked. This information will be recorded on the Certificate Revocation List. The `reasonCode` is only valid for automatic revocation requests. Manual revocation requests can use the `csrRequestorComments` parameter to tell the processing agent why the certificate is being revoked. The meaning of the `reasonCode` values are: • `0` - Unspecified • `1` - Key Compromised • `2` - CA Compromise* • `3` - Affiliation Changed • `4` - Certificate Superseded • `5` - Cessation of Operation • `6` - Certificate Hold* • `7` - (Reserved for future use)* • `8` - Remove from CRL* Values marked with an asterisk (*) are valid `reasonCode` parameters that are not used in the default `UserRevocation.html` form. These values should generally not be used for client self-revocation.

Table 24.30 Parameters Accepted by the Revocation Interface (Continued)

Parameter	Format and Description
`serialNumber`	string The serial number of the certificate to be revoked. This parameter is used for manual revocation: either a serial number or a subject name is used to identify the certificate to be revoked.
`subject`	string The subject distinguished name (DN) of the certificate to be revoked.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`templateType`	`RevocationConfirmation` `RevocationConfirmation` is the only value currently supported. This parameter is required.

Response

The default response template is revocationResult.template. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Revocation interface adds the JavaScript variables listed in the following table:

Table 24.31 Variables Returned by the Revocation Interface

Variable	Description
Generic Variable	
`revokedCerts`	The number of certificates that were revoked as a result of the request.
result.header Variables	Variables added to the header object.
`certsToUpdate`	The number of certificates that need to be updated in the publishing directory as a result of the request. Used only when directory publishing is on, indicated by the `dirEnabled` field in the header object.
`certsUpdated`	A number, less than or equal to `certsToUpdate`, indicating how many certificates have already been successfully updated in the publishing directory. Used only when directory publishing is on, indicated by the dirEnabled field in the header object.

Table 24.31 Variables Returned by the Revocation Interface (Continued)

Variable	Description
`dirEnabled`	`yes` | `no` Indicates whether directory publishing is enabled on the Certificate Manager where the revocation request was handled. May be `null` if directory publishing is not defined.
`error`	A text message indicating why the revocation request itself could not be processed. The `result.header.error` message will exist only when `result.header.revoked = no`.
`revoked`	`yes` | `pending` | `no` This field indicates the overall status of the revocation request. If the certificate could be revoked automatically, revoked will be `yes`. If the request was processed successfully, but requires manual processing by an agent, `revoked = pending`. If `revoked = no`, the request could not be processed. See `result.header.error` for the error message.
`totalRecordCount`	The total record count indicates the number of requests that were successfully processed. This number may be different from the number of certificates actually revoked. The `recordSet` array contains information for each processed revocation request (including requests that failed due to an error). The `recordSet.length` may be less than `totalRecordCount` if any certificates were already revoked.
`updateCRL`	`yes` | `no` If present and equal to `yes`, this field indicates that the Certificate Manager has attempted to update the Certificate Revocation List (CRL). Check `updateCRLSuccess` to see if the update was successful. If this field is `null` or equal to `no`, the Certificate Manger will attempt to update the CRL at the next scheduled update interval.
`updateCRLError`	This text message indicates the reason why an attempt to update the CRL has failed. This will be present if u`pdateCRL = yes` and `updateCRLSuccess = no` (or is `null`).
`updateCRLSuccess`	`yes` | `no` If `updateCRL = yes`, this field indicates whether the attempt to update the CRL has succeeded and the certificate now appears as revoked on the CRL. If this field is null or equal to `no`, see `updateCRLError`.
recordSet Variables	These fields are added to each record object in the recordSet array. They are accessed as fields of a recordSet element in the JavaScript; for example, `result.recordSet[1].error`.

Table 24.31 Variables Returned by the Revocation Interface (Continued)

Variable	Description
`error`	This text message indicates the reason that the certificate associated with this `recordSet` was processed, but could not be revoked.
`serialNumber`	Contains the serial number of the certificate represented by this `recordSet` object.

Agent Interface Reference

This section provides a detailed reference of all the service interfaces available on an agent port. For each interface, there is

- a description including the URI used, the purpose, and which agents can use it
- a list of forms that use the interface by default
- a detailed description of valid input parameters and their values
- information about the response which lists the templates used and the additional JavaScript variables available.

The following table lists the end-entity interfaces and their functions:

Table 24.32 Agent Interfaces

Interface	URI	Purpose
Approve Revocation Interface	`/ca/doRevoke` `/ra/doRevoke`	Use the id assigned to a pending request to retrieve the certificate once it has been issued.
Bulk Enrollment Interface	`/ca/bulkissuance`	Allows a process to programatically connect to the interface using SSL client authentication (with a CMS agent certificate) to submit a request and receive a certificate in response.
Display Key By Serial Number Interface	`/kra/displayBySerial`	Display information about an archived key.

Table 24.32 Agent Interfaces (Continued)

Interface	URI	Purpose
Display Key For Recovery Interface	`/kra/displayBySerialForRecovery`	Display a form for recovering a key.
Examine Recovery Interface	`/kra/examineRecovery`	Check to see if a recovery request id is valid.
Get Approval Status Interface	`/kra/getApprovalStatus`	Display the status of a key recovery operation.
Get PKCS #12 Data Interface	`/kra/getPk12`	Retrieve the PKCS #12 data containing a recovered key.
Grant Recovery Interface	`/kra/grantRecovery`	Submit an agent password to approve a key recovery.
Key Query Interface	`/kra/queryKey`	View archived keys that meet query criteria.
Key Recovery Query Interface	`/kra/queryKeyForRecovery`	Display archived keys that meet query criteria and get links to initiate recovery of these keys.
Process Certificate Request Interface	`/ca/processCertReq` `/ra/processCertReq`	Allows agents to accept or reject requests for enrollment, renewal, or revocation.
Process DRM Request Interface	`/kra/processReq`	Allows key recovery agents to view requests and change request assignments.
Process Request Interface	`/ca/processReq` `/ra/processReq`	Allows agents to view pending requests and assign them to themselves.
Recover Key By Serial Number Interface	`/kra/recoverBySerial`	Given an archive serial number, display a form for recovering the archived key.
Remove Certificate Hold Interface	`/ca/doUnrevoke` `/ra/doUnrevoke`	Remove the "on hold" revocation status of a certificate.

Requests Query Interface	`/ca/queryReq`
	`/ra/queryReq`
	`/kra/queryReq`
		View requests that match certain criteria (such as request type or status).

Select for Revocation Interface	`/ca/reasonToRevoke` `/ra/reasonToRevoke`	Revoke a set of certificates for a given reason.

Table 24.32 Agent Interfaces (Continued)

Interface	URI	Purpose
Update CRL Interface	`/ca/updateCRL` `/ra/updateCRL`	Force the Certificate Revocation List (CRL) to be updated before the scheduled update.
Update Directory Interface	`/ca/updateDir` `/ra/updateDir`	Force the publishing directory to be updated before the next scheduled update.

Approve Revocation Interface

Description

URI: `/ca/doRevoke` or `/ra/doRevoke`

Available on: Certificate Manager or Registration Manager

Function: Actually revokes a certificate or group of certificates for a given reason.

The Select for Revocation Interface is used to select a certificate or group of certificates for revocation based on some criteria. That interface returns a list of certificates using the `reasonToRevoke.template` file. The `reasonToRevoke.template` response contains a form that posts data to the Approve Revocation interface with the serial numbers and reasons to finally revoke the certificates.

Default Forms

The Approve Revocation interface is accessed through the `reasonToRevoke.template` file by default. No forms directly post data to this interface. Other forms, such as `revokeCert.html`, are used to select certificates based on query criteria; the selected certificates can then be marked for revocation by creating buttons or links that post data to the Approve Revocation interface.

Request Parameters

The following table lists the parameters that are used to revoke certificates through the Approve Revocation interface. This is an agent interface, so the HTTP POST or GET request must use SSL client authentication with a valid agent certificate.

Table 24.33 Parameters Accepted by the Approve Revocation Interface

Parameter	Format and Description
`b64eCertificate`	base-64 encoded certificate data Allows you to specify the certificate to revoke by posting its base-64 encoding to the interface.
`csrRequestorComments`	string A comment field to provide more details about why the certificates are being revoked.
`invalidityDate`	number of seconds since 1 January 1970 The time when the certificates became invalid.
`op`	`doRevoke` The only operation supported is `doRevoke`.
`requestId`	number Specifies the enrollment request id corresponding to the certificate to revoke. This allows you to post revocations to a Registration Authority.
`revocationReason`	`0` - `6` The code for the reason the certificates are being revoked. The revocation codes are: • `0` - Reason not specified • `1` - Key compromised • `2` - CA key compromised • `3` - Affiliation changes • `4` - Certificate superseded • `5` - Cessation of operation • `6` - Certificate is on hold

Table 24.33 Parameters Accepted by the Approve Revocation Interface (Continued)

Parameter	Format and Description
`revokeAll`	QUERY_FILTER For information on constructing a query filter, see Table 24.27 in the section for "List Certificates Interface" on page 958. To ensure accuracy when revoking certificates, you should use a query filter that selects each certificate by its serial number. An example value for `revokeAll` to revoke certificates with serial numbers 10 and 14 is: `(|(certRecordId=10)(certRecordId=14))`
`serialNumber`	number Specifies the serial number of a certificate to revoke.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`templateType`	string The name of the template to use for the response. The `reasonToRevoke.template` file sets this to `RevocationSuccess`.
`totalRecordCount`	number The total number of certificates selected by the `revokeAll` filter. This number can be determined from the response variables in a template used to select certificates for revocation.
`verifiedRecordCount`	number Not presently used by the interface.

Response

The default response template is `revocationResult.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Approve Revocation interface adds the JavaScript variables listed in the following table.

Table 24.34 Variables Returned by the Approve Revocation Interface

Variable	Description
result.header variables	Variables added to the header object.
`certsUpdated`	number Contains the number of certificates that were revoked from the publishing directory, if publishing is enabled (`dirEnabled = yes`).
`dirEnabled`	`yes` | `no` Indicates whether LDAP publishing is enabled on the Certificate Manager that handled the request.
`error`	message If there was an error while processing the revocation request, the error message is stored in this variable. Otherwise, the value is `null`.
`revoked`	`yes` | `pending` Indicates whether or not all certificates were successfully revoked.
`totalRecordCount`	number Indicates the total number of revocation requests that were processed as a result of the request.
`updateCRL`	`yes` | `no` Indicates whether or not the CMS server attempted to update the Certificate Revocation List (CRL). If `no` or `null`, the CRL will be updated at the next scheduled update interval. If `yes`, check `udpateCRLSuccess` to determine if the update was successful.
`updateCRLError`	message If the CMS server attempted to update the CRL and encountered an error, this variable contains the text of the error message.
`updateCRLSuccess`	`yes` | `no` If the CMS server attempted to update the CRL, this variable will indicate whether the update was successful.
result.recordSet[i] variables	Variables added to record objects in the response.

Table 24.34 Variables Returned by the Approve Revocation Interface (Continued)

Variable	Description
`error`	message If a particular certificate could not be revoked, the `error` field in its `record` object will contain an error message. If this field is `null`, the certificate was revoked successfully.
`serialNumber`	number The decimal serial number of the certificate.

Bulk Enrollment Interface

Description

URI: `/ca/bulkissuance`

Available on: Certificate Manager only

Function: The Bulk Enrollment interface allows a connection using SSL client authentication with a valid agent certificate to have a certificate issued on behalf of another entity. The entire process is automated so that a device or application with an agent certificate, the ability to do SSL client authentication, and the ability to parse and store the certificate in the response can programmatically request and receive certificates.

An application or hardware device that can also generate keys could use the Bulk Enrollment interface to generate keys, request a certificate for the public key, receive the certificate and store it (for example on a smart card to be distributed to a user).

The reply from the Bulk Enrollment interface can be just the certificate chain (in PKCS #7format), or it can be an HTML page.

Configuration Parameters

The Bulk Enrollment interface can be configured with parameters in the `CMS.cfg` configuration file.

The interface can be enabled or disabled using the `agentGateway.enableBulkInterface` parameter. The rest of the parameters configure which template file to use when a response has a given `requestStatus` code. The template configuration parameter names should be prefixed with `agentGateway.bulkissuance`. For example, `agentGateway.bulkissusance.successTemplate=/ca/bulkissuance.template`.

The following table lists the valid configuration file parameters and what they control. The file names for the template parameters are relative to the `<server_root>/<instance_id>-cert/web/agent` directory.

Table 24.35 Bulk Enrollment Interface Configuration File Parameters

Parameter	Format and Description
`agentGateway.enableBulkInterface`	`true` | `false` Enables or disables the servlet handling bulk enrollment at `/ca/bulkissuance`.
`errorTemplate`	filename The template file to use when the response `requestStatus = 6`, meaning an error occurred while processing the request.
`pendingTemplate`	filename The template file to use when the response `requestStatus = 3`, meaning the request has been deferred for manual approval.
`rejectedTemplate`	filename The template file to use when the response `requestStatus = 5`, meaning the request was rejected by a policy on the CMS server.
`successTemplate`	filename The template file to use when the response `requestStatus = 2`, meaning the certificate has been issued.
`svcpendingTemplate`	filename The template file to use when the response `requestStatus = 4`, meaning the request is pending a response from a Data Recovery Manager.
`unauthorizedTemplate`	filename The template file to use when the response `requestStatus = 1`, meaning the SSL client authentication certificate presented to authenticate the request is not a valid agent certificate.
`unexpectedErrorTemplate`	filename The template file to use when the response `requestStatus = 7`, meaning an unexpected error prevented the CMS server from processing the response.

Default Forms

No default forms use the Bulk Enrollment interface. The intent of the interface is to provide a programmatic, rather than interactive, method for enrolling entities into the PKI.

Request Parameters

The following table lists the parameters accepted by the Bulk Enrollment interface.

Note that the Bulk Enrollment interface requires SSL Client authentication with an agent certificate authorized to approve certificate requests.

Table 24.36 Parameters Accepted by the Bulk Enrollment Interface

Parameter	Format and Description
Subject Name	
`subject`	Distinguished Name (DN) string. See RFC 2253. DN to be used for the certificate subject. Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
Contact Information	
`csrRequestorName`	string Name of the entity making a request; helps identify the requestor during manual enrollment. Example: Alice Apple
`csrRequestorEmail`	string Email address of the entity making a request. May be used to send out notification when a certificate has been issued. Example: alice@example.com
`csrRequestorPhone`	string Phone number of the entity making a request. Example: 650.555.1212
`csrRequestorComments`	string Additional comments provide by the requestor on the HTML form. This field can be used if there is additional information you want to collect to help the manual enrollment.

Table 24.36 Parameters Accepted by the Bulk Enrollment Interface (Continued)

Parameter	Format and Description
Netscape Certificate Type Extensions	Parameters for setting bits in the netscape-cert-type certificate extension. See http://home.netscape.com/eng/security/comm4-cert-exts.html for details. A true value sets the bit to 1; false sets the bit to 0.
`email`	`true` | `false` Sets the S/MIME client certificate bit (bit 2).
`email_ca`	`true` | `false` Sets the S/MIME certificate issuer bit (bit 6).
`object_signing`	`true` | `false` Sets the object signing certificate bit (bit 3).
`object_signing_ca`	`true` | `false` Sets the object signing certificate issuer bit (bit 7).
`ssl_ca`	`true` | `false` Sets the SSL certificate issuer bit (bit 5).
`ssl_client`	`true` | `false` Sets the SSL client authentication certificate bit (bit 0).
`ssl_server`	`true` | `false` Sets the SSL server authentication certificate bit (bit 1).
Key Usage	Parameters for setting bits in the `keyUsage` certificate extension. A true value sets the bit to 1; false sets the bit to 0.
`crl_sign`	`true` | `false` Sets the `keyUsage` extension bit (6) indicating that the key may be used to sign Certificate Revocation Lists (CRLs).
`data_encipherment`	`true` | `false` Sets the `keyUsage` extension bit (3) indicating that the key may be used to encipher application data (as opposed to key material).
`decipher_only`	`true` | `false` Sets the `keyUsage` extension bit (8) indicating that the key may **only** be used to decipher data and keys. If this is true, keyAgreement should also be true.
`digital_signature`	`true` | `false` Sets the `keyUsage` extension bit (0) indicating that the key may be used to sign any data. This should be true for SSL client certificates, S/MIME signing certificates, and object signing certificates.
`encipher_only`	`true` | `false` Sets the `keyUsage` extension bit (7) indicating that the key may **only** be used to encipher data and keys. If this is true, keyAgreement should also be true.

Table 24.36 Parameters Accepted by the Bulk Enrollment Interface (Continued)

Parameter	Format and Description
`key_agreement`	`true` | `false` Sets the `keyUsage` extension bit (4) indicating that the key may be used to encipher and decipher keys during key agreement.
`key_certsign`	`true` | `false` Sets the `keyUsage` extension bit (5) indicating that the key may be used to sign other certificates. All CA signing certificates should set this to true.
`key_encipherment`	`true` | `false` Sets the `keyUsage` extension bit (2) indicating that the key may be used to encipher symmetric session keys. This should be true for SSL server and S/MIME encryption certificates.
`non_repudiation`	`true` | `false` Sets the keyUsage extension bit (1) indicating that the key may be used to create non-repudiable (by the signer) digital signatures. Non-repudiation service requires more infrastructure, planning, and policy than just setting this bit. Consider the ramifications before using this bit
Automated Enrollment	Parameters to configure automatic authentication for entity requests.
`authenticator`	string Specifies the name of the authentication plug-in instance to use to authenticate the entity.
`uid`	string Specifies a unique identifier passed to the authentication plug-in.
`pin`	string An optional identifying string that helps to authenticate an entity. Usually used when the Pin Generator tool has been used to populate a directory with unique identifiers for each user.
`pwd`	string Specifies the password passed to the authentication plug-in.
Other	
`certNickname`	string Specifies the nickname that should be associated with the certificate in the reply; used with Certificate Request Management Format (CRMF) requests.

Table 24.36 Parameters Accepted by the Bulk Enrollment Interface (Continued)

Parameter	Format and Description
certType	ca | CEP-Request | client | objSignClient | ra | server | other Specifies the type of certificate requested by the entity. The default is client. The certType is not associated with any certificate extensions. It may be used by policy modules to make decisions, and it may be used by a CMS server to determine how to decode the request or format the response.
challengePassword	string An optional challenge phrase or password that can be used later by the entity to revoke the certificate. This parameter is optional. If you use this, entities can use the Challenge Revocation Interface (/challenge_revocation1, page 929) with this challenge password to revoke a certificate without manual intervention and without SSL client authentication.
CRMFRequest	base-64 encoded data If requestFormat = crmf, this parameter should be used to send the base-64 encoded CRMF request.
importCAChain	true | false Used only when importCert = true. The default, if this parameter is not explicitly passed, is true. If set to true, a successful certificate request will return a PKCS #7 formatted certificate chain; if set to false, a single, DER-encoded certificate will be returned. The certificate chain includes the issued certificate and the CA (issuer) certificate.
importCert	true | false If true, and the certificate request is not deferred or rejected, the CMS server's response will be binary data with the MIME type determined by the importCertMimeType parameter. The data returned will be either a certificate or a certificate chain, based on the value of importCAChain.
importCertMimeType	MIME Type string Sets the MIME type the CMS server uses when a certificate is returned to the requestor. The default is application/x-x509-user-cert. The MIME type should be in the standard MIME type format of <type>/<subtype>.
pkcs10Request	base-64 encoded data If requestFormat = pkcs10, this parameter should be used to send the base-64 encoded certificate request.

Table 24.36 Parameters Accepted by the Bulk Enrollment Interface (Continued)

Parameter	Format and Description
`requestFormat`	`clientAuth` | `crmf` | `keygen` | `pkcs10` The value indicates the format used to submit the certificate request: • `clientAuth` - information for the new request is taken from the certificate presented by the client during SSL client authentication. • `crmf` - the certificate request is a base-64 encoded blob contained in the CRMFRequest parameter. • `keygen` - the certificate request is a base-64 encoded blob generated using the HTML `<KEYGEN>` tag. It is contained in the `subjectKeyGenInfo` parameter. • `pkcs10` - the certificate request is a base-64 encoded blob contained in the `pkcs10Request` parameter.
`subjectKeyGenInfo`	base-64 encoded data If `requestFormat=keygen`, this parameter should be used to send the base-64 encoded keygen request. To use the keygen HTML tag to cause the browser to generate the request using this parameter, the format is `<KEYGEN name="subjectKeyGenInfo">`
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

If the request parameter `importCert` is set to `true` and the certificate request is successful, the Certificate Manager will return the binary PKCS #7 certificate chain using the MIME type `application/x-x509-user-cert`. This is the most useful application of the Bulk Enrollment interface.

If `importCert` is not set to `true`, or if there is an error, the default response template is `bulkissuance.template`. Applications using the Bulk Enrollment interface should be prepared to handle the HTML output created using the template when errors occur.

The base JavaScript for responses is inserted in place of the <CMS_TEMPLATE> tag. In addition, the Bulk Enrollment interface adds the JavaScript variables listed in the following table:.

Table 24.37 Variables Returned by the Bulk Enrollment Interface

Variable	Description
result.fixed variables	Variables added to the `fixed` object.
`authorityName`	`Certificate Manager` | `Registration Manager` The name of the system that handled the request.
`certType`	`ca` | `CEP-Request` | `client` | `objSignClient` | `ra` | `server` | `other` The type of certificate returned. This value is the same as the `certType` value passed to the interface in the request.
`errorDescription`	string A message providing more details about the error described in `errorDetails`. This variable is only present if an error occurred while processing the request.
`errorDetails`	string A message explaining the error that occurred while processing the enrollment request. This variable is only present if an error occurred while processing the request.
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same port.
`port`	number The port number that was used to service the request.
`requestId`	number A unique number assigned by the CMS server to this request. This is especially useful for pending requests since there is no unique certificate serial number yet assigned.

Table 24.37 Variables Returned by the Bulk Enrollment Interface (Continued)

Variable	Description
`requestStatus`	number A code indicating the current status of the request: • 1 (Unauthorized): The request specified a value for an authenticator to perform an automated enrollment, and the authenticator did not authorize the request. • 2 (Success): Processing the request was successful and a certificate has been issued. If `importCert` was set to `true`, the response will include code (from the `EnrollSuccess.template`) to import the certificate into the browser making the request. Otherwise, the response is only a success message. • 3 (Pending): The request has been successfully processed by the CMS server and added to a queue for approval by an agent. If the request has been submitted to another Certificate Manager or Data Recovery Manager and is currently pending in the queue for that service, the response template will be `GenSvcPending.template` instead of `GenPending.template`. • 4 (Reserved): Not currently used. • 5 (Rejected): The request was rejected during policy processing. • 6 (Error): An error occurred when the CMS server processed the request. The error may be the result of missing or improperly formatted parameters. • 7 (Exception): An unknown or unexpected error occurred when the CMS server processed the request.
`scheme`	`http` | `https` The protocol that was used to make the request. Use this along with host and port to make sure any new requests to the end-entity port use the correct scheme.
`unexpectedError`	string A message explaining the exception or unexpected error that occurred.
result.recordSet[i] variables	Variables added to each `record` object. Each `record` object is added as an element of the `recordSet` array. Multiple records may be returned if more than one certificate was generated as a result of the request. Dual-key requests (for example, if the request parameter `requestFormat = crmf`) may return two certificates if the request is successfully processed and approved.
`base64Cert`	string The newly issued certificate in base-64 encoded format. This string includes the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.

Table 24.37 Variables Returned by the Bulk Enrollment Interface (Continued)

Variable	Description
`certFingerprint`	string A string of hexadecimal numbers separated by colons that represent the certificate fingerprints. There are three substrings: one each for the MD2, MD5, and SHA1 fingerprint. Each fingerprint begins with the hash algorithm name and a colon, and ends with a newline (\n).
`certPrettyPrint`	string A long text string that shows all of the certificate data in a human readable form.
`policyMessage`	string If the request was rejected by policy processing on the CMS server, this variable will contain a message explaining why.
`serialNo`	number The serial number (in decimal) of the newly issued certificate.

Display Key By Serial Number Interface

Description

URI: `/kra/displayBySerial`

Available on: Data Recovery Manager

Function: Displays information in human-readable form about a single archived key.

The Display Key By Serial Number interface is typically used within a form that lists keys to display detailed information about a selected key.

Default Forms

The Display Key By Serial Number interface is used in the `queryKey.template` file. Each key in the list of keys satisfying the query has a button the user can press to see the key in detail. This button submits data to the Display Key By Serial Number interface.

Request Parameters

The following table lists the parameters accepted by the Display Key By Serial Number interface.

Table 24.38 Parameters Accepted by the Display Key By Serial Number Interface

Parameter	Format and Description
`op`	`displayBySerial` Specifies the operation to perform. The only valid value is `displayBySerial`.
`serialNumber`	number The serial number of the key to display. Note that this is the DRM serial number, not the serial number from a certificate.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `displayBySerial.template` (note that the template in the `web/agents/kra` directory differs significantly from the template for Certificate Managers, Registration Managers, or end entities; those forms are used to display certificates, not keys).

The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Display Key By Serial Number interface adds the JavaScript variables listed in the following table:

Table 24.39 Variables Returned by the Display Key By Serial Number Interface

Variable	Description
result.header variables	Variables added to the header object.
`archivedBy`	user ID The user ID of the agent that processed the key archival request.
`archivedOn`	number of seconds since 1 January 1970 The time when the key was stored in the archive (for completed Data Recovery Manager requests).

Table 24.39 Variables Returned by the Display Key By Serial Number Interface (Continued)

Variable	Description
`keyAlgorithm`	OID string The object identifier (OID) used by the archived key corresponding to this request (Data Recovery Manager requests). For example, the OID for an RSA encryption key is `"1.2.840.113549.1.1.1."`
`keyLength`	number The number of bits in the archived key (Data Recovery Manager requests).
`op`	`displayBySerial` Indicates the operation that was requested.
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to an archived encryption key (Data Recovery Manager requests only). Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`publicKey`	string A string of two-digit hexadecimal numbers separated by colon. Each number represents a byte in the public key corresponding to the private key in the archive.
`serviceURL`	string Indicates the URI that was used to request this form. By default, this will always be `"/kra/displayBySerial"`.
`state`	`VALID` | `INVALID` The current status of the key corresponding to the request.

Display Key For Recovery Interface

Description

URI: `/kra/displayBySerialForRecovery`

Available on: Data Recovery Manager

Function: Displays a form for recovering a key.

The Display Key For Recovery interface is typically used in the list returned by the Key Recovery Query Interface. The purpose of the interface is to retrieve information about a specific key (based on its DRM serial number) and present a form to collect the rest of the information required to start the recovery process. The response template, `displayBySerialForRecovery.template`, will render a form that uses the Recover Key By Serial Number Interface.

Default Forms

The Display Key For Recovery interface is used in the `queryKeyForRecovery.template` file. Each key in the list of keys satisfying the query has a button the user can press to start the recovery process. This button submits data to the Display Key For Recovery interface.

Request Parameters

The following table lists the parameters accepted by the Display Key For Recovery interface.

Table 24.40 Parameters Accepted by the Display Key For Recovery Interface

Parameter	Format and Description
`op`	`displayBySerial` Specifies the operation to perform. The only valid value is `displayBySerial`.
`publicKeyData`	base-64 encoded certificate data This optional parameter allows you to pass the certificate corresponding to the key to revoke to the interface. The certificate will be required to recover the key, and passing it here allows the certificate to be automatically filled in on the resulting form.
`serialNumber`	number The serial number of the key to display. Note that this is the DRM serial number, not the serial number from a certificate.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `displayBySerialForRecovery.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Display Key For Recovery interface adds the JavaScript variables listed in the following table:

Table 24.41 Variables Returned by the Display Key For Recovery Interface

Variable	Description
result.header variables	Variables added to the header object.
`archivedBy`	user ID The user ID of the agent that processed the key archival request.
`archivedOn`	number of seconds since 1 January 1970 The time when the key was stored in the archive (for completed Data Recovery Manager requests).
`keyAlgorithm`	OID string The object identifier (OID) used by the archived key corresponding to this request (Data Recovery Manager requests). For example, the OID for an RSA encryption key is `"1.2.840.113549.1.1.1."`
`keyLength`	number The number of bits in the archived key.
`noOfRequiredAgents`	number Indicates the number of authorized agents who must approve the request before the key can be recovered.
`op`	`displayBySerial` Indicates the operation that was requested.
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to an archived encryption key (Data Recovery Manager requests only). Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`publicKey`	string A string of two-digit hexadecimal numbers separated by colon. Each number represents a byte in the public key corresponding to the private key to be archived.

Table 24.41 Variables Returned by the Display Key For Recovery Interface (Continued)

Variable	Description
`recoveryID`	number A unique identification number assigned to each recovery request when it gets created.
`serviceURL`	string Indicates the URI that was used to request this form. By default, this will always be `"/kra/displayBySerial"`.
`state`	`VALID` | `INVALID` The current status of the key corresponding to the request.

Examine Recovery Interface

Description

URI: `/kra/examineRecovery`

Available on: Data Recovery Manager

Function: Checks to see if a recovery request identification number is valid.

The Examine Recovery interface is an intermediate interface that validates a request identification number. The interface takes a request identification number and makes sure that it is associated with a valid recovery request on the CMS server. If the request number is not valid, an error is returned (using the `GenError.template` file). If the request number is valid, the interface returns a template that by default creates a form for posting a user ID and password to the Grant Recovery Interface.

Default Forms

The Examine Recovery interface is used by the `grantRecovery.html` form in the Data Recovery Manager web directory (`web/agents/kra`). The form accepts a recovery id and posts it to the Examine Recovery interface, which returns either a form for granting the recovery or an error message (if the request id is invalid).

Request Parameters

The following table lists the parameters accepted by the Examine Recovery interface.

Table 24.42 Parameters Accepted by the Examine Recovery Interface

Parameter	Format and Description
`op`	`examineRecovery` The only operation supported by the Examine Recovery interface is `examineRecovery`.
`recoveryID`	number The unique identification number assigned to the recovery request.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The `GenError.template` file is used to return any error messages. Successful requests use the `examineRecovery.template` file. Since the interface is used to validate a request id, the `examineRecovery.template` is used by default to create a form to submit a user ID and password to the Grant Recovery Interface to approve that request.

The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Examine Recovery interface adds the JavaScript variables listed in the following table:

Table 24.43 Variables Returned by the Examine Recovery Interface

Variable	Description
result.header variables	Variables added to the header object.
`op`	`examineRecovery` Indicates the operation requested. Only `examineRecovery` is supported.

Table 24.43 Variables Returned by the Examine Recovery Interface (Continued)

Variable	Description
`recoveryID`	number The unique request identification number that was passed to the interface in the request. Since the `examineRecovery.template` has been returned, the `recoveryID` has been verified to correspond to a valid request.
`serialNumber`	number The serial number of the key in the archive.
`serviceURL`	`/kra/examineRecovery` The URL that was used to access the Examine Recovery interface.

Get Approval Status Interface

Description

URI: `/kra/getApprovalStatus`

Available on: Data Recovery Manager

Function: Displays the status of a recovery operation.

The Get Approval Status interface accepts a recovery request number and returns the status of the request. The response includes the number of agents required to approve the recovery and the number that have already granted approval. While a request is pending, agents can use the Grant Recovery Interface to submit user IDs and passwords. The Examine Recovery Interface can be used as an intermediate step to make sure that there is a valid request corresponding to a request identification number (this is the default behavior of the Authorize Recovery link from the Data Recovery Manager gateway).

Default Forms

The Get Approval Status interface requires a valid request id number, so it is used by default only in the response template for the Recover Key By Serial Number Interface. The Recover Key By Serial Number Interface returns a request id number for pending (non-local) requests. In the default response, this request id number is used in a form that polls the Get Approval Status interface using a `<META HTTP-EQUIV="Refresh">` tag to make continuous

requests until the recovery is completed. The page looks at the `result.recordSet` array to see how many agents have approved the request so far.

Request Parameters

The following table lists the parameters accepted by the Get Approval Status interface.

Table 24.44 Parameters Accepted by the Get Approval Status Interface

Parameter	Format and Description
`recoveryID`	number The unique identification number assigned to the recovery request.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template depends on the status of the request. While the request is pending, the `getApprovalStatus.template` file is used. Once the request is completed the `finishRecovery.template` file is used. By default, these templates are identical except for the `<META HTTP-EQUIV="Refresh" CONTENT="5">` tag in `getApprovalStatus.template`, which causes the page to refresh itself every 5 seconds.

The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Get Approval Status interface adds the JavaScript variables listed in the following table:

Table 24.45 Variables Returned by the Get Approval Status Interface

Variable	Description
result.header variables	Variables added to the header object.
`errorDetails`	string If an error occurred while processing the request, this variable contains an error message.

Table 24.45 Variables Returned by the Get Approval Status Interface (Continued)

Variable	Description
`noOfRequiredAgents`	number The number of agents required to supply passwords before the key can be recovered. Compare to `result.recordSet.length`, which indicates how many agents have supplied valid passwords so far.
`recoveryID`	number The unique number assigned to this recovery request.
`serialNumber`	number The serial number of the key in the archive.
`status`	`complete` | `null` Once the required number of agents have supplied valid passwords, the status becomes `complete` and the `finishRecovery.template` is used for the response. Until the `status` is `complete`, `result.header.status` is not included in the response.
result.recordSet[i] variables	Variables added to `record` objects in the response.
`agentName`	string The user ID of an agent that has supplied a valid password to approve the recovery.

Get PKCS #12 Data Interface

Description

URI: `/kra/getPk12`

Available on: Data Recovery Manager

Function: Retrieves the PKCS #12 data containing a recovered key and certificate.

The Get PKCS #12 Data interface is used to retrieve the PKCS #12 blob containing a recovered key and its associated certificate. The PKCS #12 data is encrypted using the password supplied to the Recover Key By Serial Number Interface.

Default Forms

No default forms use the Get PKCS #12 Data interface. The `finishRecovery.template`, `getApprovalStatus.template`, and `recoverBySerial.template` files all embed links to the Get PKCS #12 Data interface that are displayed if the recovery has been granted (if `result.header.status = "complete"`).

Request Parameters

The Get PKCS #12 Data interface requires only a recovery request ID. This parameter is set in the HTML form and passed to the interface using HTTP.

Table 24.46 Parameters Accepted by the Get PKCS #12 Data Interface

Parameter	Format and Description
`recoveryID`	number The unique identification number assigned to the recovery request.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

If the recovery has been granted, the response is to return a PKCS #12 blob with the MIME type `application/x-pkcs12`. This blob is encrypted with a password (supplied when the recovery request was initiated) and contains the key and corresponding certificate.

If the recovery has not yet been granted, the default response template is `finishRecovery.template`. See the Response section for the "Get Approval Status Interface" on page 994 for details on the JavaScript returned.

Grant Recovery Interface

Description

URI: `/kra/grantRecovery`

Available on: Data Recovery Manager

Function: Submits a password to approve a key recovery.

The Grant Recovery interface is used by agents to submit their passwords to authorize a key recovery. Key recovery requires a certain number of authorized agents submit passwords before the key can be recovered. Each agent uses the Grant Recovery interface to submit a password until the minimum number of passwords has been collected.

Default Forms

The Grant Recovery interface is used by the `examineRecovery.template` result template by default. This form is returned when a request ID has been verified as valid, so it ensures that the Grant Recovery interface will be called with a valid request id.

Request Parameters

The following table lists the parameters accepted by the Grant Recovery interface.

Table 24.47 Parameters Accepted by the Grant Recovery Interface

Parameter	Format and Description
`agentID`	string The key recovery agent id used by the agent making the request. Agent ids for key recovery are configured independently of agent user IDs, and so may be different.
`agentPWD`	string The password corresponding to the key recovery agent user ID submitted with `agentID`.

Table 24.47 Parameters Accepted by the Grant Recovery Interface (Continued)

Parameter	Format and Description
`op`	`grantRecovery` The only operation supported by the Grant Recovery interface is `grantRecovery`.
`recoveryID`	number The unique identification number assigned to the recovery request.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `grantRecovery.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Grant Recovery interface adds the JavaScript variables listed in the following table:

Table 24.48 Variables Returned by the Grant Recovery Interface

Variable	Description
result.header variables	Variables added to the header object.
`agentID`	string The key recovery agent id whose password was submitted.
`op`	`grantRecovery` Indicates the operation requested. Only `grantRecovery` is supported.
`recoveryID`	number The unique request identification number that was passed to the interface in the request.
`serialNumber`	number The serial number of the key in the archive.
`serviceURL`	`/kra/grantRecovery` The URL that was used to access the Grant Recovery interface.

Key Query Interface

Description

URI: `/kra/queryKey`

Available On: Data Recovery Manager only.

Function: Retrieves a set of archived keys based on a flexible query specification.

The Key Query interface allows you to build query criteria much like an LDAP query. Criteria can be combined using logical AND or OR for flexibility. You can match keys in the archive based on the DRM serial number, the DN of the key owner, the certificate containing the corresponding public key, or the agent that archived the key.

The interface returns the public keys corresponding to the archived keys that match the query criteria.

Default Forms

The Data Recovery Manager form `searchKey.html` uses the Key Query interface. The form allows the agent to specify any of the valid query parameters: serial number range, key owner DN, public key certificate, or user ID of the archiving agent.

Request Parameters

The following table lists the parameters accepted by the Key Query interface.

The `queryFilter` parameter must be a valid query filter. The syntax and valid query parameters are too complex to describe in the parameter table. Details about valid parameters and values for query filters are in a separate table following the parameters.

Table 24.49 Parameters Accepted by the Key Query Interface

Parameter	Format and Description
`maxCount`	number Specifies the maximum number of keys to display on each page returned. If more than `maxCount` keys match the search criteria, each page will have controls to see the next or previous page of results.
`op`	`queryKey` The only operation supported by the List Certificates interface is `queryKey`.

`queryFilter`	`([<OP>]<FILTER>[<FILTER>...])`
	Details about building query filters are provided in the next table.
	The `queryFilter` must be enclosed in parentheses.
	The `<OP>` argument, required if there is more than one `<FILTER>`, specifies how the filters that follow should be logically evaluated:
	• `&` (ampersand) means that the filters should be linked by a logical AND: all filters must evaluate to true for the expression to match a key.
	• `|` (pipe) means that the filters should be linked by a logical OR: if at least one filter evaluates to true, the expression matches a key.
	Any number of filters can be concatenated within any set of parentheses.
	An example filter is
	`(|(keyArchivedBy=drmagent1)(&(keySerialNumber>=5)(keySerialNumber<=35)))`
	This filter matches any key that was archived by `drmagent1` or keys with serial numbers between 5 and 35 in the archive.

`querySentinel`	number The `querySentinel` indicates which record out of the total matching set should be the first displayed on the resulting output page. For example, if `totalRecordCount = 15` and `maxCount = 5`, set `querySentinel` to 6 to show the 6th through 10th element of the matching set.

Table 24.49 Parameters Accepted by the Key Query Interface (Continued)

Parameter	Format and Description
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`totalRecordCount`	number The total number of keys in the archive that match the `queryFilter`. This number is returned by the interface in the initial response. This can be posted in subsequent calls to prevent the CMS server from calculating a number. In this way every page can tell the user the absolute size of the matching set, rather than have the size of the set appear to decrease on each next page of data.

The following table describes the parameter names that are valid for constructing query filters, and the range of valid values that can be used with the parameter. The parameters can be combined using parentheses and logical operators (as described in the previous table) to construct query filters of arbitrary complexity.

In a filter, the parameter name is compared to the expression value using one of the relational operators = (matches), < (less than), <= (less than or equal to), > (greater than), or >= (greater than or equal to). Some expressions (such as `keySerialNumber`) accept the asterisk (*) as a wildcard to match 0 or more

characters; for example, `"keyOwnerName=CN=*,*OU=Engineering*"` matches `"CN=jsmith, OU=Engineering, O=Organization1, C=US"` and `"CN=jdoe, OU=Engineering, O=Organization2, C=CA."`

Table 24.50 Key Query queryFilter Parameters

Parameter	Expression Values
`keyArchivedBy`	Value: user ID of an agent This matches the user ID of the agent that approved archival. For example: `(keyArchivedBy=drmagent1)`
`keySerialNumber`	Value: serial number of a key This matches the serial number assigned by the Data Recovery Manager to a key when it is archived. Note that this differs from the serial number of the certificate corresponding to the key. Use the asterisk (*) wildcard to match any or partial serial numbers. For example: `(&(keySerialNumber>=1)(keySerialNumber<=10))`

`keyOwnerName`	Value: distinguished name appearing as the subject of the certificate corresponding to the key.
	This parameter matches the DN (taken from the certificate) stored with the key during archival. Use the wildcard (`*`) to match parts of a DN. If users are asked to enter a DN and it is likely that they will not remember all of the data, use the wildcard to allow partial matches.
	For example:
	`<INPUT TYPE=TEXT NAME=partialDN>`
	`form.queryFilter.value = "(keyOwnerName=*" + partialDN + "*)";`

`publicKey`	Value: `x509cert#`<base-64 encoded certificate>
	The query will find the archived key corresponding to the public key in the certificate. See `searchKey.html` for an example. The certificate in the proper format can be retrieved from a Certificate Manager using an interface such as the Display Certificate By Serial Number Interface.
	For example:
	`(publicKey=x509cert#"-----BEGIN CERTIFICATE-----`
	`\nMII\"BFK364J978nmnJK89yjha\"asFDSJ973-----END CERTIFICATE-----`
	`")`

Response

The default response template is `queryKey.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Key Query interface adds the JavaScript variables listed in the following table.

Table 24.51 Variables Returned by the Key Query Interface

Variable	Description
result.header variables	Variables added to the header object.
`archiverName`	Distinguished Name (DN) string. See RFC 2253. The subject name on the Data Recovery Manager's signing certificate.
`op`	`queryKey` Indicates the operation that was requested.
`queryFilter`	query string Contains the query string that was used to select keys from the archive. See the previous section, "Request Parameters" on page 1000, for details on how `queryFilter` is constructed. The queryFilter should be reused to query for more keys when there are more records than displayed on the current page.
`querySentinel`	number The number of the first record shown on this page of output. This is an index relative to the `totalRecordCount`. For example, if `querySentinel = 6`, `totalRecordCount = 8`, and `maxCount = 5`, then the 6th through 8th records that matched the query will be displayed.
`templateName`	string Indicates the name of the template that was used to display the response. By default it is `queryKey.template`, but it can be changed with the `templateName` request parameter.
`totalRecordCount`	number The total number of records that match the query criteria (`queryFilter`). This may be more than the number of records currently displayed, which is limited by `fixed.maxCount`.
result.fixed variable	Variable added to the `fixed` object.
`maxCount`	number The maximum number of records that can be displayed on a single page of output (taken from the request parameter `maxCount`).
result.recordSet[i] variables	Variables added to `record` objects in the response.

Table 24.51 Variables Returned by the Key Query Interface (Continued)

Variable	Description
`archivedBy`	user ID The user ID of the agent that processed the key archival request.
`archivedOn`	number of seconds since 1 January 1970 The time when the key was stored in the archive (for completed Data Recovery Manager requests).
`keyAlgorithm`	OID string The object identifier (OID) used by the archived key corresponding to this request (Data Recovery Manager requests). For example, the OID for an RSA encryption key is `"1.2.840.113549.1.1.1."`
`keyLength`	number The number of bits in the archived key (Data Recovery Manager requests).
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to an archived encryption key (Data Recovery Manager requests only). Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`publicKey`	string A string of two-digit hexadecimal numbers separated by colon. Each number represents a byte in the public key corresponding to the private key to be archived.
`serialNumber`	number A unique identification number that identifies a key in the archive. This differs from the certificate serial number.
`state`	`VALID | INVALID` The current status of the key corresponding to the request.

Key Recovery Query Interface

Description

URI: `/kra/queryKeyForRecovery`

Available On: Data Recovery Manager only.

Function: Retrieves a set of archived keys, for the purpose of recovering them, based on a flexible query specification.

The Key Recovery Query interface allows you to build query criteria much like an LDAP query. Criteria can be combined using logical AND or OR for flexibility. You can match keys in the archive based on the DRM serial number, the DN of the key owner, the certificate containing the corresponding public key, or the agent that archived the key.

The interface returns the public keys corresponding to the archived keys that match the query criteria. The list of keys in the response will each have a "Recover" button that allows the key to be recovered.

Default Forms

The Data Recovery Manager form `recoverKey.html` uses the Key Recovery Query interface. The form allows the agent to specify any of the valid query parameters: serial number range, key owner DN, public key certificate, or user ID of the archiving agent.

Request Parameters

The following table lists the parameters accepted by the Key Recovery Query interface.

The `queryFilter` parameter must be a valid query filter. The syntax and valid query parameters are too complex to describe in the parameter table. The parameter is identical to the `queryFilter` request parameter for the Key Query Interface; refer to “Key Query Interface” on page 1000 for complete details on building queries.

Table 24.52 Parameters Accepted by the Key Recovery Query Interface

Parameter	Format and Description
`maxCount`	number Specifies the maximum number of keys to display on each page returned. If more than `maxCount` keys match the search criteria, each page will have controls to see the next or previous page of results.
`op`	`queryKey` The only operation supported by the List Certificates interface is `queryKey`.

Table 24.52 Parameters Accepted by the Key Recovery Query Interface (Continued)

Parameter	Format and Description
`publicKeyData`	base-64 encoded certificate The certificate containing the public key associated with a key to be matched in the archive. The `publicKeyData` should include the "`-----BEGIN CERTIFICATE-----`" header and "`-----END CERTIFICATE-----`" footer.

`queryFilter`	`([<OP>]<FILTER>[<FILTER>...])`
	Details about building query filters are provided in the next table.
	The `queryFilter` must be enclosed in parentheses.
	The `<OP>` argument, required if there is more than one `<FILTER>`, specifies how the filters that follow should be logically evaluated:
	• `&` (ampersand) means that the filters should be linked by a logical AND: all filters must evaluate to true for the expression to match a key.
	• `|` (pipe) means that the filters should be linked by a logical OR: if at least one filter evaluates to true, the expression matches a key.
	Any number of filters can be concatenated within any set of parentheses.
	An example filter is
	`(|(keyArchivedBy=drmagent1)(&(keySerialNumber>=5)(keySerialNumber<=35)))`
	This filter matches any key that was archived by `drmagent1` or keys with serial numbers between 5 and 35 in the archive.

`querySentinel`	number The `querySentinel` indicates which record out of the total matching set should be the first displayed on the resulting output page. For example, if `totalRecordCount = 15` and `maxCount = 5`, set `querySentinel` to 6 to show the 6th through 10th element of the matching set.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`totalRecordCount`	number The total number of keys in the archive that match the queryFilter. This number is returned by the interface in the initial response. This can be posted in subsequent calls to prevent the CMS server from calculating a number. In this way every page can tell the user the absolute size of the matching set, rather than have the size of the set appear to decrease on each next page of data.

Response

The default response template is `queryKeyForRecovery.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Key Recovery Query interface adds the JavaScript variables listed in the following table.

Table 24.53 Variables Returned by the Key Recovery Query Interface

Variable	Description
result.header variables	Variables added to the header object.
`archiverName`	Distinguished Name (DN) string. See RFC 2253. The subject name on the Data Recovery Manager's signing certificate.
`op`	`queryKey` Indicates the operation that was requested.
`queryFilter`	query string Contains the query string that was used to select keys from the archive. See the previous section, "Request Parameters" on page 1000, for details on how `queryFilter` is constructed. The queryFilter should be reused to query for more keys when there are more records than displayed on the current page.
`querySentinel`	number The number of the first record shown on this page of output. This is an index relative to the `totalRecordCount`. For example, if `querySentinel = 6`, `totalRecordCount = 8`, and `maxCount = 5`, then the 6th through 8th records that matched the query will be displayed.
`templateName`	string Indicates the name of the template that was used to display the response. By default it is `queryKey.template`, but it can be changed with the `templateName` request parameter.
`totalRecordCount`	number The total number of records that match the query criteria (`queryFilter`). This may be more than the number of records currently displayed, which is limited by `fixed.maxCount`.
result.fixed variable	Variable added to the `fixed` object.
`maxCount`	number The maximum number of records that can be displayed on a single page of output (taken from the request parameter `maxCount`).

Table 24.53 Variables Returned by the Key Recovery Query Interface (Continued)

Variable	Description
result.recordSet[i] variables	Variables added to `record` objects in the response.
`archivedBy`	user ID The user ID of the agent that processed the key archival request.
`archivedOn`	number of seconds since 1 January 1970 The time when the key was stored in the archive (for completed Data Recovery Manager requests).
`keyAlgorithm`	OID string The object identifier (OID) used by the archived key corresponding to this request (Data Recovery Manager requests). For example, the OID for an RSA encryption key is `"1.2.840.113549.1.1.1."`
`keyLength`	number The number of bits in the archived key (Data Recovery Manager requests).
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to an archived encryption key (Data Recovery Manager requests only). Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`publicKey`	string A string of two-digit hexadecimal numbers separated by colon. Each number represents a byte in the public key corresponding to the private key to be archived.
`serialNumber`	number A unique identification number that identifies a key in the archive. This differs from the certificate serial number.
`state`	`VALID` | `INVALID` The current status of the key corresponding to the request.

Process Certificate Request Interface

Description

URI: `/ca/processCertReq` or `/ra/processCertReq`

Available on: Certificate Manager or Registration Manager

Function: Agents can use the Process Certificate Request interface to accept, reject, or cancel requests to sign, renew, or revoke requests.

The Process Request Interface is used to assign pending requests to agents. The Process Certificate Request interface is used by an agent to act on requests owned by the agent (or not yet owned).

The interface is called once the sequence number of a pending request is known, so forms that use the interface are usually embedded in a response template from an interface that can list pending requests (for example, the Requests Query Interface). The list of pending requests will also include the `requestType` variable to distinguish between signing, revocation, and renewal requests.

Default Forms

No default forms use the Process Certificate Request Interface directly. The `listRequests.html` form calls the Requests Query Interface which returns data using the `processReq.template` when pending requests are selected.

The `processReq.template` will create a link to the Process Certificate Request Interface for all requests with `status == pending`. Additional JavaScript in that template is used to add parameters to the request to the Process Certificate Request interface (such as the validity period for enrollments and renewals).

Request Parameters

The following table lists the parameters accepted by the Process Request Interface.

The agent interface requires SSL client authentication, so information about the agent can be gleaned from the certificate used to authenticate and does not need to be passed in parameters.

Table 24.54 Parameters Accepted by the Process Certificate Request Interface

Parameter	Format and Description
`addExts`	base-64 encoded certificate extensions Specifies any additional certificate extensions that should be added. There is sample code in the `cms_sdk/samples/exttools` directory that can be used to generate the base-64 encoding of some extensions; the code can be modified to generate other extensions.
`certTypeEmail`	`yes` | `no` Specifies whether to set the netscape certificate extension S/MIME client certificate bit (bit 2).
`certTypeEmailCA`	`yes` | `no` Specifies whether to set the netscape certificate extension S/MIME certificate issuer bit (bit 6).
`certTypeObjSigning`	`yes` | `no` Specifies whether to set the netscape certificate extension object signing certificate bit (bit 3).
`certTypeObjSigningCA`	`yes` | `no` Specifies whether to set the netscape certificate extension object signing certificate issuer bit (bit 7).
`certTypeSSLCA`	`yes` | `no` Specifies whether to set the netscape certificate extension SSL certificate issuer bit (bit 5).
`certTypeSSLClient`	`yes` | `no` Specifies whether to set the netscape certificate extension SSL client authentication certificate bit (bit 0).
`certTypeSSLServer`	`yes` | `no` Specifies whether to set the netscape certificate extension SSL server authentication certificate bit (bit 1).
`checkPubKeyUniqueness`	`yes` | `no` Specifies whether the CMS server should ensure that the new certificate's public key is unique.
`checkValidityNesting`	`yes` | `no` Specifies whether the CMS server should check to make sure that the certificate does not expire later than the CA's signing certificate.

Table 24.54 Parameters Accepted by the Process Certificate Request Interface (Continued)

Parameter	Format and Description
`csrRequestorEmail`	string The email address of the party making the original signing request.
`csrRequestorName`	string The real name of the party making the original signing request.
`csrRequestorPhone`	string The phone number of the party making the original signing request.
`grantCMAgentPrivilege`	`yes` | `no` Specifies whether the new certificate will be trusted as a Certificate Manager agent.
`grantDRMAgentPrivilege`	`yes` | `no` Specifies whether the new certificate will be trusted as a Data Recovery Manager agent.
`grantRMAgentPrivilege`	`yes` | `no` Specifies whether the new certificate will be trusted as a Registration Manager agent.
`grantTrustedManagerPrivilege`	`yes` | `no` Specifies whether the new certificate will be an SSL certificate used by a server that is trusted. For example, set this to `yes` when you issue the SSL server certificate for a new Registration Manager.
`grantUID`	user ID The user ID that will be associated with the agent certificate.
`seqNum`	number The sequence number of the request that is being acted upon. When requests are listed from the Requests Query Interface, their sequence numbers are returned from the CMS server in the `result.header.seqNum` field.
`signatureAlgorithm`	`MD5withRSA` | `SHA1withDSA` | `SHA1withRSA` Specifies the signing algorithm that should be used to sign a newly issued certificate. The CA signing key must match the key type (RSA or DSA) of the selected algorithm.
`subject`	Distinguished Name (DN) string. See RFC 2253. DN to be used for the certificate subject. Example: CN=Alice Apple, UID=alice, OU=People, O=Example, C=US
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Table 24.54 Parameters Accepted by the Process Certificate Request Interface (Continued)

Parameter	Format and Description
`toDo`	`accept` | `cancel` | `clone` | `reject` Specifies the action to take on the request.
`validityLength`	number of seconds The length of time, in seconds, for which the newly issued certificate will be valid. The following list shows the approximate number of seconds in some common time intervals: • 1 day: 86,400 • 7 days: 604,800 • 14 days: 1,209,600 • 30 days: 2,592,000 • 180 days: 15,552,000 • 365 days: 31,536,000 • 540 days: 46,665,000 • 730 days: 63,072,000

Response

The default response template is `processCertReq.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Process Certificate Request interface adds the JavaScript variables listed in the following table.

Table 24.55 Variables Returned by the Process Certificate Request Interface

Variable	Description
result.header variables	Variables added to the header object.
`assignedTo`	The user ID of the agent to whom the request is currently assigned. Compare to `callerName` to see if the agent viewing the request is the current owner.
`authorityid`	`ca` | `ra` The type of authority that handled the request: `ca` for Certificate Manager or `ra` for Registration Manager.
`callerName`	The user ID of the agent who is viewing the request. This data is determined from the SSL client certificate presented to the agent interface.

Table 24.55 Variables Returned by the Process Certificate Request Interface (Continued)

Variable	Description
certsUpdated	number The number of certificates that were updated in the publishing directory if publishing is enabled (if result.header.dirEnabled = yes).
certType	ca | CEP-Request | client | objSignClient | ra | server | other Specifies the type of certificate request that was acted upon. The certType is not associated with any certificate extensions. It may be used by policy modules to make decisions, and it may be used by a CMS server to determine how to decode the request or format the response.
clonedRequestId	number The sequence number (or request ID) of the newly cloned request, if the action toDo = clone.
createdOn	number of seconds since 1 January 1970 The time when the certificate request was created on the CMS server.
dirEnabled	yes | no Indicates whether or not LDAP publishing is enabled on the CMS server. If yes, then certsUpdated will indicate how many certificates were updated in the directory as a result of the certificate processing action.
errorDetails	message A more detailed description of any processing errors.
errors	message A message explaining any errors that may have occurred.
ext_email	true | false Indicates whether or not the Netscape certificate extension S/MIME bit (bit 2) is set in the certificate or request that was processed.
ext_email_ca	true | false Indicates whether or not the Netscape certificate extension S/MIME CA bit (bit 6) is set in the certificate or request that was processed.
ext_object_signing	true | false Indicates whether or not the Netscape certificate extension object signing bit (bit 3) is set in the certificate or request that was processed.
ext_object_signing_ca	true | false Indicates whether or not the Netscape certificate extension object signing CA bit (bit 7) is set in the certificate or request that was processed.

Table 24.55 Variables Returned by the Process Certificate Request Interface (Continued)

Variable	Description
`ext_ssl_ca`	`true` | `false` Indicates whether or not the Netscape certificate extension SSL CA bit (bit 5) is set in the certificate or request that was processed.
`ext_ssl_client`	`true` | `false` Indicates whether or not the Netscape certificate extension SSL client bit (bit 0) is set in the certificate or request that was processed.
`ext_ssl_server`	`true` | `false` Indicates whether or not the Netscape certificate extension SSL server bit (bit 1) is set in the certificate or request that was processed.
`grantError`	`SUCCESS` | error message If there were any errors processing a request for an agent certificate, this field will have an error message. If `grantError = SUCCESS`, the agent was created successfully.
`grantPrivilege`	string Indicates the groups to which the new agent or certificate has been added. If there is more than one group, the group names will be separated by the text `" and "` in the string. Valid group names are: • `Certificate Manager Agents` • `Data Recovery Manager Agents` • `Registration Manager Agents` • `Trusted Managers` (for SSL certificates used by trusted servers)
`grantUID`	user ID The user name assigned to the agent certificate that was processed.
`localca`	`yes` | `no` Indicates whether the CMS server that processed the request is a Certificate Manager (the CMS server is not a Registration Manager).
`localkra`	`yes` | `no` Indicates whether the CMS server that processed the request also hosts a Data Recovery Manager.
`requestType`	`enrollment` | `getCAChain` | `getCertificates` | `getRevocationInfo` | `renewal` | `revocation` | `unrevocation` The `requestType` returns the type of request that was made to the interface returning this template.

Table 24.55 Variables Returned by the Process Certificate Request Interface (Continued)

Variable	Description
`serialNumber`	number The serial number of the certificate that was processed. This might be a newly issued certificate or a newly revoked certificate.
`seqNum`	number The unique sequence number assigned to the request by the Certificate Manager or Registration Manager.
`signatureAlgorithm`	OID The object identifier (OID) string corresponding to the algorithm used to sign a newly issued certificate. For example, `"1.2.840.113549.1.1.4"` is the OID for MD5 with RSA signing.
`signatureAlgorithmName`	`MD5withRSA` | `SHA1withDSA` | `SHA1withRSA` The name token associated with the signature algorithm whose OID is stored in `signatureAlgorithm`.
`status`	`pending` | `complete` Indicates whether the request is complete or if further action is required. Responses from the Process Certificate Request interface will return a `status` of `complete` unless an error occurred.
`subject`	Distinguished Name (DN) string. See RFC 2RFC 2253. The DN appearing in the certificate subject field. Example DN: CN=Alice Apple, UID=alice, OU=People, O=Example, C=US
`subjectPublicKey`	string of hexadecimal numbers The actual public key in the certificate that was processed. The string is a sequence of two-digit hexadecimal numbers separated by colons: `"A1:3F:23:90:D7\n"` Each pair of digits represents a byte or octet in the public key.
`subjectPublicKeyInfo`	string A string indicating the public key algorithm and certificate signing algorithm. For example, `"RSA - 1.2.840.113549.1.1.1"` for a certificate with an RSA key signed using MD5 with RSA.
`toDo`	`accept` | `cancel` | `clone` | `reject` Indicates the action that was taken to produce this response (that is, this is the same as the value of the `toDo` request parameter).
`updatedBy`	user ID The user ID of the agent that updated the request. In the case of a response from the Process Certificate Request interface, this is the user ID of the agent that made the request.

Table 24.55 Variables Returned by the Process Certificate Request Interface (Continued)

Variable	Description
`updatedOn`	number of seconds since 1 January 1970 The time that the request was made to the Process Certificate Request interface.
`validityLength`	number of seconds The length of time, in seconds, for which the newly issued certificate will be valid. The following list shows the approximate number of seconds in some common time intervals: • 1 day: 86,400 • 7 days: 604,800 • 14 days: 1,209,600 • 30 days: 2,592,000 • 180 days: 15,552,000 • 365 days: 31,536,000 • 540 days: 46,665,000 • 730 days: 63,072,000
result.recordSet[i] variables	Variables added to record objects in the response.
`ext_prettyprint`	string A representation of any extensions present in the certificate in human-readable form.
`serialNumber`	number The decimal serial number of the certificate.

Process DRM Request Interface

Description

URI: `/kra/processReq`

Available on: Data Recovery Manager

Function: This interface allows an agent to view a request or assign the request to himself.

The Process DRM Request interface is slightly different from the Process Request Interface used by Certificate Managers and Registration Managers.

Default Forms

The Process DRM Request interface is not used in any default forms. The interface requires the sequence number of an archival request, so it is used in templates that list requests (with their sequence numbers) to render buttons that allow an agent to view or change the assignment of a request.

Request Parameters

The following table lists the parameters accepted by the Process DRM Request interface.

The agent interface requires SSL client authentication, so information about the agent can be gleaned from the certificate used to authenticate and does not need to be passed in parameters.

Table 24.56 Parameters Accepted by the Process DRM Request Interface

Parameter	Format and Description
`doAssign`	`yes` | `no` Specifies whether to assign the request to the agent using the interface. If this parameter is `no`, `null`, or not supplied, the request will be displayed, but not assigned to the agent.
`moreComments`	string Specifies additional comments to be stored with the request. Comments may be useful for future reference or to provide data to another agent that needs to process the request.
`op`	`processReq` The only operation supported is `processReq`.
`overrideAssignment`	`yes` | `no` Specifies whether or not to override an existing assignment. If a request is already assigned to another agent, an agent must use `doAssign = yes` and `overrideAssignment = yes` to assume assignment of the request. If the request is not assigned, `overrideAssignment` is not required.
`requestorEmail`	email address The email address of the entity requesting archival or recovery, if the information is available.

Table 24.56 Parameters Accepted by the Process DRM Request Interface (Continued)

Parameter	Format and Description
`requestorName`	string The name of the entity requesting archival or recovery, if the information is available.
`requestorPhone`	string The phone number of the entity requesting archival or recovery, if the information is available.
`seqNum`	number The sequence number of a pending request to assign or redisplay.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `processReq.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Process DRM Request interface adds the JavaScript variables listed in the following table.

Table 24.57 Variables Returned by the Process DRM Request Interface

Variable	Description
result.header variables	Variables added to the header object.
`archivedBy`	user ID The user ID of the Data Recovery Agent that authorized the archival. This is sent only for completed archival requests.
`archivedOn`	seconds since 1 January 1970 The time when the key was stored in the archive. Sent only for completed archival requests.
`callerName`	user ID The user ID of the agent who is viewing the request. This data is determined from the SSL client certificate presented to the agent interface.
`createdOn`	seconds since 1 January 1970 The time when the request was created.

Table 24.57 Variables Returned by the Process DRM Request Interface (Continued)

Variable	Description
`keyAlgorithm`	OID The object identifier (OID) of the algorithm used by the key in the archive request.
`keyLength`	number The number of bits in the key in the archive request.
`localca`	`yes` | `no` Indicates whether the Certificate Manager associated with the Data Recovery Manager is located on the same host.
`localkra`	`yes` | `no` Indicates whether the Data Recovery Manager is located on the same host as the Process DRM Request interface. For this interface, the value is always `yes`.
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject name on the certificate corresponding to the key to be archived. This appears only for archival requests.
`publicKey`	string A string of two-digit hexadecimal numbers separated by colon. Each number represents a byte in the public key corresponding to the private key to be archived.
`requestType`	`enrollment` | `recovery` Indicates whether the request was made to archive (`enrollment`) or recover a key.
`serialNumber`	number A unique identification number that identifies a key in the archive. This differs from the certificate serial number and also from the request identifier (`seqNum`).
`seqNum`	number The sequence number assigned to the request by the Data Recovery Manager.
`state`	`VALID` | `INVALID` The current status of the key corresponding to the request.
`status`	`pending` | `complete` Only requests that have `status == pending` need to use the Process DRM Request interface to assign the request to an agent. Requests that are `complete` can only be displayed.

Table 24.57 Variables Returned by the Process DRM Request Interface (Continued)

Variable	Description
`updatedBy`	user ID Indicates the user ID of the agent that last changed the status of the request.
`updatedOn`	seconds since 1 January 1970 The time when the request status was last changed.

Process Request Interface

Description

URI: `/ca/processReq` or `/ra/processReq`

Available on: Certificate Manager or Registration Manager

Function: Changes the assignment of a pending certificate request to an agent.

This is an agent interface and requires SSL client authentication with a valid agent certificate. The Process Request Interface can be used to assign a certificate request (identified by a sequence number) to the agent user ID associated with the certificate presented for authentication or to assign the request to nobody (remove any existing assignment).

Default Forms

The Process Request Interface is not used in any default forms. The interface requires the sequence number of a certificate request, so it is used in templates that list pending requests (with their sequence numbers) to render buttons that allow an agent to change the assignment of a request.

Request Parameters

The following table lists the parameters accepted by the Process Request interface.

The agent interface requires SSL client authentication, so information about the agent can be gleaned from the certificate used for authentication and does not need to be passed in parameters.

Table 24.58 Parameters Accepted by the Process Request Interface

Parameter	Format and Description
`doAssign`	`reassignToMe` | `reassignToNobody` | `assignToMe` | `null` Specifies how to change the agent assigned to a pending request. Use `reassignToMe` if the request has already been assigned to another agent or `assignToMe` if the request has not been assigned (or has just been assigned with `reassignToNobody`). If the interface is used with no `doAssign` parameter, the request in question is redisplayed using the `processReq.template`.
`seqNum`	number The sequence number of a pending request to assign or redisplay.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is processReq.template. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag.

In addition, the Process Request interface adds the JavaScript variables listed in the following table. The table lists only the variables in the template related to the Process Request interface. The template includes more variables related to the status and origin of the request; these are documented in other interfaces that use the template.

Table 24.59 Variables Returned by the Process Request Interface

Variable	Description
result.header variables	Variables added to the header object.
`assignedTo`	The user ID of the agent to whom the request is currently assigned. Compare to `callerName` to see if the agent viewing the request is the current owner.

Table 24.59 Variables Returned by the Process Request Interface (Continued)

Variable	Description
`callerName`	The user ID of the agent who is viewing the request. This data is determined from the SSL client certificate presented to the agent interface.
`requestType`	`enrollment` | `getCAChain` | `getCertificates` | `getRevocationInfo` | `renewal` | `revocation` | `unrevocation` The `requestType` returns the type of request that was made to the interface returning this template. The Process Request Interface is used to assign pending requests, so only requests for `enrollment`, `renewal`, `revocation`, and `unrevocation` would need a link to the Process Request Interface.
`seqNum`	number The sequence number assigned to the request by the Certificate Manager or Registration Manager.
`status`	`pending` | `complete` Only requests that have `status == pending` need to use the Process Request interface to assign the request to an agent.

Recover Key By Serial Number Interface

Description

URI: `/kra/recoverBySerial`

Available on: Data Recovery Manager

Function: Displays a form for recovering a key.

Once a key has been selected from the archive, the Recover Key By Serial Number Interface can be used to start the recovery operation. This interface accepts the serial number of an archived key and optionally the user ID and passwords of recovery agents.

The recovery operation can be completed by one call to the Recover Key By Serial Number Interface if the proper number of recovery agent user IDs and passwords are provided. In this case, a PKCS #12 blob is returned with the recovered key and an associated public key certificate.

The recovery operation can also be deferred if the agent passwords cannot be immediately provided. In this case, the response from the interface is a new form that polls the Get Approval Status Interface until the passwords are entered by agents (from other locations) and the PKCS #12 blob can be delivered.

Default Forms

The Recover Key By Serial Number Interface typically follows a request to the Display Key For Recovery Interface. The result of the Display Key For Recovery Interface request is to display a key from the archive using the `displayBySerialForRecovery.template` file. This template provides a form for making a request to the Recover Key By Serial Number Interface to initiate the recovery operation for the selected key.

Request Parameters

The following table lists the parameters accepted by the Recover Key By Serial Number interface.

Table 24.60 Parameters Accepted by the Recover Key By Serial Number Interface

Parameter	Format and Description
`cert`	base-64 encoded PKCS #7 certificate You must supply the certificate containing the public key corresponding to the key in the archive. Be sure to include the "-----BEGIN CERTIFICATE-----" header and "-----END CERTIFICATE-----" footer.
`localAgents`	`yes` | `no` Specifies whether agent user IDs and passwords have been entered locally (that is, submitted with this request), or will be entered remotely. If `localAgents = no`, agents will have to access the Examine Recovery interface with the request id and enter their passwords.
`nickname`	string Specifies an optional nickname for the certificate that will be returned in the PKCS #12 blob when the key is recovered.
`op`	`recoverBySerial` Specifies the operation to perform. The only valid value is `recoverBySerial`.

Table 24.60 Parameters Accepted by the Recover Key By Serial Number Interface (Continued)

Parameter	Format and Description
`p12password`	string Specifies a password used to protect the recovered key. When the PKCS #12 blob containing the key is returned, this password will be required to decrypt the data.
`p12passwordAgain`	string This parameter serves as a quality check; the string must match the value of `p12password`.
`pwd<n>`	string Specifies the password for agent `<n>`, where `<n>` is replaced by a sequence number (beginning with 0). If this is a local recovery operation, user IDs (`uid<n>`) and passwords must be supplied for the number of agents required to authorize a recovery. For example, if two agents are required, a local recovery operation requires `uid0`, `pwd0`, `uid1`, and `pwd1` parameters.
`serialNumber`	number The serial number of the key to recover. Note that this is the DRM serial number, not the serial number from a certificate.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`uid<n>`	string Specifies the user ID for agent number `<n>`, if this is a local operation. See the description for `pwd<n>`.

Response

If the request was for a local recovery (`localAgents = yes` in the request), and the recovery is successful, the response will be the binary PKCS #12 blob containing the key and certificate. The MIME type of the response will be `application/x-pkcs12`.

If the request was not local or if there was an error, the default response template is `recoverBySerial.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. If there are no errors, the default template includes JavaScript code that uses `window.location` to immediately replace the returned page with a call to the Get Approval Status Interface.

In addition, the Recover Key By Serial Number interface adds the JavaScript variables listed in the following table:

Table 24.61 Variables Returned by the Recover Key By Serial Number Interface

Variable	Description
result.fixed variables	Variables added to the fixed object
`host`	string The fully qualified domain name of the CMS server that processed the request. This allows the resulting template to construct forms that post data to the same interface using the same host and port.
`port`	number The port number that was used to service the request.
`scheme`	`http` | `https` The protocol that was used to make the request.
result.header variables	Variables added to the header object.
`errorDetails`	string If an error occurred while processing the request, this variable contains an error message.
`recoveryID`	number The unique number assigned to this recovery request. If agents need to approve the request, they will supply this number to the Examine Recovery Interface.

Remove Certificate Hold Interface

Description

URI: `/ca/doUnrevoke` or `/ra/doUnrevoke`

Available on: Certificate Manager or Registration Manager agent ports.

Function: Changes the status of a certificate that has been put on hold so that it is no longer considered revoked.

A certificate can be temporarily rendered invalid --or "put on hold"-- by revoking it with a revocation reason code of 6. If subsequent analysis reveals that the certificate does not need to be permanently revoked, it can be made valid again using the Remove Certificate Hold Interface.

Only agents with valid SSL client certificates can use the Remove Certificate Hold interface.

Default Forms

There are no default forms that use this interface. The Remove Certificate Hold Interface is usually accessed after a list of certificates currently on hold has been generated. In the response template that displays the list, certificates on hold can be rendered with a "Remove Certificate Hold" button next to their listing. The `displayByCert.template` and `queryCert.template` create buttons that use the Remove Certificate Hold Interface.

Request Parameters

The following table lists the parameters accepted by the Remove Certificate Hold interface.

Table 24.62 Parameters Accepted by the Remove Certificate Hold Interface

Parameter	Format and Description
`op`	`doUnrevoke` Specifies the operation to perform. The valid value for the Remove Certificate Hold interface is "`doUnrevoke`."
`serialNumber`	number The serial number (in decimal or hexadecimal) of the certificate to revoke.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `unrevocationResult.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Remove Certificate Hold Interface adds the JavaScript variables listed in the following table:

Table 24.63 Variables Returned by the Remove Certificate Hold Interface

Variable	Description
result.header variables	Variables added to the header object.
`dirEnabled`	`yes` | `no` Indicates whether LDAP publishing is enabled on the Certificate Manager that handled the request.
`dirUpdated`	`yes` | `no` If directory publishing is enabled, this indicates whether or not the directory was updated with the new certificate status.
`error`	message If there was an error while processing the unrevocation request, the error message is stored in this variable. Otherwise, the value is `null`.
`publishCRLError`	message If the CMS server attempted to publish the CRL and encountered an error, this variable contains the text of the error message.
`publishCRLSuccess`	`yes` | `no` If the CMS server attempted to publish the CRL to a directory, this variable will indicate whether the update was successful.
`serialNumber`	number The decimal serial number of the certificate.
`unrevoked`	`yes` | `no` | `pending` Indicates whether or not all certificates were successfully unrevoked.
`updateCRL`	`yes` | `no` Indicates whether or not the CMS server attempted to update the Certificate Revocation List (CRL). If `no` or `null`, the CRL will be updated at the next scheduled update interval. If `yes`, check `udpateCRLSuccess` to determine if the update was successful.

Table 24.63 Variables Returned by the Remove Certificate Hold Interface (Continued)

Variable	Description
`updateCRLError`	message If the CMS server attempted to update the CRL and encountered an error, this variable contains the text of the error message.
`updateCRLSuccess`	`yes` | `no` If the CMS server attempted to update the CRL, this variable will indicate whether the update was successful.

Requests Query Interface

Description

URI: `/ca/queryReq` or `/ra/queryReq` or `/kra/queryReq`

Available on: Certificate Manager, Registration Manager, or Data Recovery Manager

Function: Lists requests that have been made to a given server.

The interface can return all requests, or subsets based on request status (pending, complete, etc.) and request type (enrollment, renewal, etc.).

Default Forms

The `listRequests.html` form uses the Requests Query Interface. This form can be found in the Certificate Manager, Registration Manager, and Data Recovery Manager web directories. The `listRequests.html` form presents menus for choosing the request type and status as well as a field for setting the lowest request id to return.

Since the number or records returned by the interface may be more than the user wants to see on one page, the response template (`queryReq.template`) may have a button to retrieve more records that also uses the Requests Query Interface.

Request Parameters

The following table lists the parameters that are used to view requests through the Requests Query Interface. This is an agent interface, so the HTTP POST or GET request must use SSL client authentication with a valid agent certificate.

Table 24.64 Parameters Accepted by the Requests Query Interface

Parameter	Format and Description
`maxCount`	number Specifies the maximum number of requests to display on a page. If more than `maxCount` requests match the criteria, the response template can include code to request more records from the Requests Query interface.
`op`	`queryReq` Specifies the operation to perform. For the Requests Query interface, this should be `queryReq`.
`reqState`	`showAll` | `showCancelled` | `showCompleted` | `showInService` | `showRejected` | `showWaiting` Specifies the status of requests to show.
`reqType`	`archival` | `enrollment` | `getCAChain` | `getCertificates` | `getCRL` | `getRevocationInfo` | `recovery` | `renewal` | `revocation` | `showAll` | `unrevocation` Specifies the type of request to show.
`seqNumFrom`	number Specifies the lowest request identification number to retrieve. This parameter is useful when the number of requests is more than `maxCount` and another page of data can be requested: set `seqNumFrom` to one more than the last request displayed on the current page and repost the request.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`totalRecordCount`	number The total number of requests that match the criteria. This is, of course, not known until at least one page of requests have been retrieved. Requests to see more data can pass this number along so that the total number of matching requests can be displayed on every page (otherwise the total would decrease as subsequent requests used higher values for `seqNumFrom`). If this parameter is not passed, the total will be calculated on the CMS server, and the response table will include the total in the `result.header.totalRecordCount` variable.

Response

The default response template is `queryReq.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag.

For each request returned, a record object is created. Some of the record object fields listed below may not apply to some requests; for example, a pending or rejected enrollment request will have no certificate subject.

The Requests Query interface adds the JavaScript variables listed in the following table.

Table 24.65 Variables Returned by the Requests Query Interface

Variable	Description
result.header variables	Variables added to the header object.
`authorityId`	ca | kra | ra The type of server that generated the list of requests: ca for Certificate Manager, kra for Data Recovery Manager, or ra for Registration Manager.
`currentRecordCount`	number The number of request records displayed on the current page of output.
`error`	message If there was an error while processing the revocation request, the error message is stored in this variable. Otherwise, the value is `null`.
`querySentinel`	number A tracking number that indicates the default number of records to retrieve on the next page of output. This number is the lesser of the maxCount requested and the total number of records left in the result set.
`requestingUser`	user ID The user ID of the agent that requested the list of requests.
`totalRecordCount`	number The total number of request records that matched the criteria.
result.fixed variables	Variables added to the fixed object.
`maxCount`	number The maximum number of records to display on a page. This is taken from the `maxCount` request parameter.

Table 24.65 Variables Returned by the Requests Query Interface (Continued)

Variable	Description
`reqState`	`showAll` | `showCancelled` | `showCompleted` | `showInService` | `showRejected` | `showWaiting` Indicates the request state parameter that was used to generate the list.
`reqType`	`archival` | `enrollment` | `getCAChain` | `getCertificates` | `getCRL` | `getRevocationInfo` | `recovery` | `renewal` | `revocation` | `showAll` | `unrevocation` Indicates the request type parameter that was used to generate the response.
`seqNumFrom`	number The lowest request id number displayed in the current list of requests. This value is taken from the `seqNumFrom` request parameter.
result.recordSet[i] variables	Variables added to record objects in the response.
`archivedBy`	user ID The user ID of the agent that processed the key archival request.
`archivedOn`	number of seconds since 1 January 1970 The time when the key was stored in the archive (for completed Data Recovery Manager requests).
`assignedTo`	user ID The user ID of the agent currently assigned to the request. If no agent is assigned, `assignedTo` will be `null`.
`callerName`	user ID The user ID of the agent that requested this list of requests.
`certType`	`ca` | `CEP-Request` | `client` | `objSignClient` | `ra` | `server` | `other` Indicates the type of certificate.
`createdOn`	seconds since 1 January 1970 Indicates the time when the request was created.
`keyAlgorithm`	OID string The object identifier (OID) used by the archived key corresponding to this request (Data Recovery Manager requests). For example, the OID for an RSA encryption key is `"1.2.840.113549.1.1.1."`
`keyLength`	number The number of bits in the archived key (Data Recovery Manager requests).

Table 24.65 Variables Returned by the Requests Query Interface (Continued)

Variable	Description
`ownerName`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to an archived encryption key (Data Recovery Manager requests only). Example: `CN=Alice Apple, UID=alice, OU=People, O=Example, C=US`
`requestType`	`enrollment` | `getCAChain` | `getCertificates` | `getRevocationInfo` | `renewal` | `revocation` | `unrevocation` Indicates the type of request for this record.
`serialNumber`	number The unique identification number for the request on a Data Recovery Manager. Registration Managers and Certificate Managers use `seqNum`.
`seqNum`	number The request identification number for this request. The request ID is unique on any instance of a server. The seqNum is used for Registration Manager and Certificate Manager Requests. Data Recovery Manager requests are indexed by the `serialNumber` field.
`state`	`VALID` | `INVALID` For Data Recovery Manager requests, the current state of the archived key.
`status`	`cancelled` | `complete` | `rejected` | `waiting` The current status of the request.
`subject`	Distinguished Name (DN) string. See RFC 2253. The subject entry on the certificate corresponding to this request (if there is one).
`updatedBy`	user ID The user ID of the agent who last updated this request.
`updatedOn`	seconds since 1 January 1970 The time when this request was last updated.

Select for Revocation Interface

Description

URI: `/ca/reasonToRevoke` or `/ra/reasonToRevoke`

Available on: Certificate Manager or Registration Manager

Function: Displays a set of certificates matching a query filter to be revoked.

The Select for Revocation Interface uses a query filter to select certificates. The response template lists these certificates in a form that allows them to be revoked using the Approve Revocation Interface.

Default Forms

By default forms that use the Select for Revocation Interface are embedded as buttons on certificate lists returned from the List Certificates Interface accessed through an agent port (`/ca/listCerts` or `/ra/listCerts`). These response are rendered using the `queryCert.template` discussed in the List Certificates Interface section.

Request Parameters

The following table lists the parameters that are used to select certificates through the Select for Revocation Interface. This is an agent interface, so the HTTP POST or GET request must use SSL client authentication with a valid agent certificate.

Table 24.66 Parameters Accepted by the Select For Revocation Interface

Parameter	Format and Description
`revokeAll`	QUERY_FILTER For information on constructing a query filter, see Table 24.27 in the End-Entity Interfaces "List Certificates Interface" section. To ensure accuracy when revoking certificates, you should use a query filter that selects each certificate by its serial number. An example value for `revokeAll` to revoke certificates with serial numbers 10 and 14 is: `(|(certRecordId=10)(certRecordId=14))`

Table 24.66 Parameters Accepted by the Select For Revocation Interface (Continued)

Parameter	Format and Description
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`totalRecordCount`	number The total number of certificates to select for revocation. This value is simply passed through and shows up in the response template as `result.header.totalRecordCount`.

Response

The default response template is `reasonToRevoke.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Select for Revocation interface adds the JavaScript variables listed in the following table.

Table 24.67 Variables Returned by the Select For Revocation Interface

Variable	Description
result.header variables	Variables added to the header object.
`caSerialNumber`	number The decimal serial number of the Certificate Authority's signing certificate.
`revokeAll`	QUERY_FILTER The query filter that was used in the request to select the certificates that appear in this response. An example value for `revokeAll` to revoke certificates with serial numbers 10 and 14 is: `(|(certRecordId=10)(certRecordId=14))`
`totalRecordCount`	number The value of totalRecordCount specified in the request.
`verifiedRecordCount`	number The number of revocable certificate that were actually selected based on the query criteria.
result.recordSet[i] variables	Variables added to record objects in the response.

Table 24.67 Variables Returned by the Select For Revocation Interface (Continued)

Variable	Description
`error`	message If a particular certificate could not be revoked, the `error` field in its `record` object will contain an error message. If this field is `null`, the certificate was revoked successfully.
`serialNumber`	number The decimal serial number of the certificate.
`subject`	string The subject distinguished name of the certificate. For example, `"CN=Jane Doe, UID=jdoe, OU=Users, O=Organization, ST=California, C=US."`
`validNotAfter`	number of seconds since 1 January, 1970 The date when the certificate expires. See the description for `issuedOn` for details on date values.
`validNotBefore`	number of seconds since 1 January, 1970 The date when the certificate became valid. See the description for `issuedOn` for details on date values.

Update CRL Interface

Description

URI: `/ca/updateCRL` or `/ra/updateCRL`

Available on: Certificate Manager and Registration Manager agent ports.

Function: Certificate Revocation Lists (CRLs) are automatically updated on a regular basis. If necessary, this interface can be used to force an update to the CRL.

Default Forms

The form `updateCRL.html`, available in the CA agent and RA agent directories, uses the Update CRL Interface. The form allows the user to select a signing algorithm for the CRL.

Request Parameters

The following table lists the parameters accepted by the Update CRL interface.

Table 24.68 Parameters Accepted by the Update CRL Interface

Parameter	Format and Description
`crlIssuingPoint`	`MasterCRL` Specifies the issuing point maintained by the CMS server handling the CRL update. In the default case, the only issuing point for all CRL information is the master CRL. If other issuing points have been configured in the Certificate Manager's configuration file, you can use the token used to define the issuing point for this parameter.
`signatureAlgorithm`	`MD5withRSA` | `SHA1withDSA` | `SHA1withRSA` Specifies the signing algorithm that should be used to sign the updated CRL. The CA signing key must match the key type (RSA or DSA) of the selected algorithm.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.

Response

The default response template is `updateCRL.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. This simple template uses the crlPublished variable to display either a success or failure message.

In addition, the Update CRL interface adds the JavaScript variables listed in the following table:

Table 24.69 Variables Returned by the Update CRL Interface

Variable	Description
result.header variables	Variables added to the header object.
`crlPublished`	`Success` | `Failure` Indicates whether or not the CRL was successfully updated and signed.
error	message A message explaining why the CRL update failed.

Update Directory Interface

Description

URI: `/ca/updateDir` or `/ra/updateDir`

Available on: Certificate Manager and Registration Manager.

Function: If enabled, the publishing directory is automatically updated on a regular basis. If necessary, this interface can be used to force new information to be published to the directory.

The interface allows all new information or just selected subsets (for example, only updated expired certificate information) to be published.

Default Forms

The form `updateDir.html`, available in the CA agent and RA agent directories, uses the Update Directory Interface.

Request Parameters

The following table lists the parameters accepted by the Update Directory interface.

Table 24.70 Parameters Accepted by the Update Directory Interface

Parameter	Format and Description
`expiredFrom`	number The low end of the range of serial numbers of expired certificates to be updated in the directory. For no lower bound, set this to null or omit the parameter.
`expiredTo`	number The high end of the range of serial numbers of expired certificates to be updated in the directory. For no upper bound, set this to null or omit the parameter.
`revokedFrom`	number The low end of the range of serial numbers of revoked certificates to be updated in the directory. For no lower bound, set this to null or omit the parameter.

Table 24.70 Parameters Accepted by the Update Directory Interface (Continued)

Parameter	Format and Description
`revokedTo`	number The high end of the range of serial numbers of revoked certificates to be updated in the directory. For no upper bound, set this to null or omit the parameter.
`templateName`	string Filename relative to the template directory (`web/ee`, `web/agent/ca`, `web/agent/kra`, or `web/agent/ra`) of a file to use as the response template. This template will be used for any response, overriding default template settings.
`updateAll`	`yes` | `no` Whether to update all information in the directory ("`yes`") or only to update selected categories ("`no`"). If "`no`," be sure to set one of `updateCA`, `updateCRL`, `updateExpired`, `updateRevoked`, or `updateValid` to "`yes`."
`updateCA`	`yes` | `no` Whether or not to update the Certificate Manager's signing certificate in the directory.
`updateCRL`	`yes` | `no` Whether or not to update the certificate revocation list (CRL) in the directory. Any new updates since the last automatic or manual update will be published.
`updateExpired`	`yes` | `no` Whether or not to remove certificates from the directory that have expired since the last update. If you want to restrict the range of certificates (by serial number), specify values for `expiredFrom`, `expiredTo`, or both.
`updateRevoked`	`yes` | `no` Whether or not to remove certificates from the directory that have been revoked since the last update. If you want to restrict the range of certificates (by serial number), specify values for `revokedFrom`, `revokedTo`, or both.
`updateValid`	`yes` | `no` Whether or not to publish certificates that have been issued (or otherwise become valid) since the last update to the directory. If you want to restrict the range of certificates (by serial number), specify values for `validFrom`, `validTo`, or both.
`validFrom`	number The low end of the range of serial numbers of valid certificates to be updated in the directory. For no lower bound, set this to null or omit the parameter.
`validTo`	number The high end of the range of serial numbers of valid certificates to be updated in the directory. For no upper bound, set this to null or omit the parameter.

Response

The default response template is `updateDir.template`. The base JavaScript for responses is inserted in place of the `<CMS_TEMPLATE>` tag. In addition, the Update Directory interface adds the JavaScript variables listed in the following table.

A variable will not be added (it will have a `null` value) if it does not apply; for example, if updating the CRL was not requested, the `crlPublished` variable will not be present (`crlPublished == null` will evaluate to `true`).

Table 24.71 Variables Returned by the Update Directory Interface

Variable	Description
result.header variables	Variables added to the header object.
`caCertError`	string A message explaining why the CA certificate could not be published to the directory, if there was an error.
`caCertPublished`	`Success` | `Failure` If updating the CA certificate was requested, this variable will indicate whether the update was successful or not. See `caCertError` for an error message in case of `Failure`.
`crlError`	string A message explaining why the CRL could not be published to the directory, if there was an error.
`crlPublished`	`Success` | `Failure` If updating the CRL was requested, this variable will indicate whether the update was successful or not. See `crlError` for an error message in case of `Failure`.
`expiredCertsError`	string A message explaining why the expired certificates could not be removed from the directory, if there was an error.
`expiredCertsUnpublished`	`Success` | `Failure` If removing expired certificates was requested, this variable will indicate whether the update was successful or not. See `expiredCertsError` for an error message in case of `Failure`.
`revokedCertsError`	string A message explaining why the revoked certificates could not be removed from the directory, if there was an error.

Table 24.71 Variables Returned by the Update Directory Interface (Continued)

Variable	Description
`revokedCertsUnpublished`	`Success` | `Failure` If removing revoked certificates was requested, this variable will indicate whether the update was successful or not. See `revokedCertsError` for an error message in case of `Failure`.
`validCertsError`	A message explaining why new certificates could not be published to the directory, if there was an error.
`validCertsPublished`	`Success` | `Failure` If publishing new certificates was requested, this variable will indicate whether the update was successful or not. See `validCertsError` for an error message in case of `Failure`.

9

Logs

Chapter

25

Introduction to Logs

Netscape Certificate Management System (CMS) creates log files that record events related to its activities, such as administration, communications using any of the protocols the server supports, and various other processes employed by all the subsystems the server manages.

This chapter identifies various logs maintained by Certificate Management System and describes them in detail. The chapter has the following sections:

Logs Maintained by Certificate Management System

While Certificate Management System is running, it keeps a log of information and error messages on all the components it manages. These messages are broadly categorized into three separate logs and are maintained in three separate log files, as listed in Table 25.1.

During installation, Certificate Management System automatically creates the required log files in your local file system. The server creates common system, error, and audit files for all components that were installed together, and it logs messages to these files. For example, if you installed a Certificate Manager and a Data Recovery Manager together, you will find log messages for both the subsystems in the same log file.

Table 25.1 Types of logs maintained by Certificate Management System

Log type	Description
System	This log records information about requests to the server (all HTTP and HTTPS requests) and the responses from the server. Information recorded in this log includes the IP address of the client machine that accessed the server, operations performed (for example, search, add, edit), and the result of the access (for example, the number of entries returned). This log is on by default. For more information, see "Configuring System Logs" on page 1058 and "Monitoring System Logs" on page 1065.
Error	This log contains the error messages the server has encountered (HTTP errors and errors with the certificate service). This log is on by default. For more information, see "Configuring Error Logs" on page 1061 and "Monitoring Error Logs" on page 1068.
Audit	This log records messages specific to the certificate service—messages such as certificate requests, certificate renewal and revocation requests, and CRL publication—and enables you to detect any unauthorized access or activity. This log is on by default. For more information, see "Configuring Audit Logs" on page 1063 and "Monitoring Audit Logs" on page 1070.

Services That Are Logged

All major components and protocols (or services) of Certificate Management System log messages to log files. Table 25.2 lists services that are logged by default. If you want to view messages logged by a specific service, you can customize log settings accordingly. For details, see "Monitoring Logs" on page 1064.

Table 25.2 Services logged by Certificate Management System

Service	Description
All	Specifies logged events related to all the services.
Registration Authority	Specifies logged events related to the Registration Manager.
Certificate Authority	Specifies logged events related to the Certificate Manager.
Key Recovery Authority	Specifies logged events related to the Data Recovery Manager.
HTTP	Specifies logged events related to the HTTP activity of the server.
Database	Specifies logged events related to this server's activity with the internal database.
Authentication	Specifies logged events related to this server's activity with the authentication module.
Administration	Specifies logged events related to this server's administration activities—that is, HTTPS communication between the CMS window and Certificate Management System.
LDAP	Specifies logged events related to this server's activity with the LDAP directory (used for publishing certificates and CRLs).
Request Queue	Specifies logged events related to the request queue activity of this server.
ACLs	Specifies logged events related to access control lists.
User and Group	Specifies logged events related to users and groups managed by this server.
Others	Specifies logged events related to other activities of this server, such as command-line utilities and other processes.

Log Levels (Message Categories)

For identification and filtering purposes, events logged by all CMS-supported services are classified into various categories. These are listed in Table 25.3. Each category represents messages that are of the same or a similar nature or that belong to a specific functional area. A particular log, for example the error log, can record entries that fall under one or more of these categories.

In the CMS configuration, each message category corresponds to a specific log level. Log levels are represented by numbers (digits) 1 to 6, each digit indicating the level of logging to be performed by the server—that is, how detailed the logging should be.

- A higher priority level (a larger digit) means less detail because only events of high priority are logged.
- A lower priority level (a smaller digit) means greater detail because more kinds of events are recorded in the log file.

Table 25.3 Classification of log entries or messages

Log level	Message category	Description
0	Debugging	These messages contain debugging information.
1	Informational (default selection for audit log)	These messages provide general information about the state of Certificate Management System. For example, status messages such as "Certificate Management System initialization complete" and "Request for operation succeeded" fall into this category.
2	Warning	These messages are warnings only and do not indicate any failure in the normal operation of the server.
3	Failure (default selection for system and error logs)	These messages indicate errors and failures that prevent the server from operating normally. Examples of messages that fall into this category include failures to perform a certificate service operation ("User authentication failed" or "Certificate revoked") and unexpected situations that can cause irrevocable errors ("The server cannot send back the request it processed for a client through the same channel the request came from the client").
4	Misconfiguration	These messages indicate that a misconfiguration in the server is causing an error.

Table 25.3 Classification of log entries or messages (Continued)

Log level	Message category	Description
5	Catastrophic failure	These messages indicate that because of an error, the service cannot continue running.
6	Security-related events	These messages identify occurrences that affect the security of the server (for example, "Privileged access attempted by user with revoked or unlisted certificate").

You can use log levels to filter log entries based on the severity of an event. By default, a level 3 (Failure) is set for all services.

Important The log level is additive—that is, specifying a value of 3 causes levels 4, 5, and 6 to be logged. Log data can be voluminous, especially at lower (more verbose) logging levels. Make sure that the host machine has sufficient disk space for all the log files. It is also important to define your logging level, log rotation, log expiration, and server-backup policies appropriately so that all the log files are backed up and the host system doesn't get overloaded; otherwise, you may lose information.

Log File Locations

For quick access, all the log files—system, error, and audit—are maintained in your local file system. Make sure that your storage capacity is sufficient for all your log files. A log file has the following default location:

```
<server_root>/cert-<instance_id>/logs
```

You can change the default location for logs by modifying it in the configuration. For details, see "Log Parameters in the Configuration File" on page 1058.

Log File Naming Conventions

All log files created by Certificate Management System use one or the other of two naming conventions. There is one naming convention for active log files and one for rotated log files.

Active Log File Naming Convention

All active log files created by Certificate Management System use an identical naming convention. The name of an active log file is in the form `<log_type>.log`, where `<log_type>` specifies the log file type—whether it is system, error, or audit.

For example, an active error log file would be named `error.log`.

Rotated Log File Naming Convention

All rotated log files created by Certificate Management System use an identical naming convention. When Certificate Management System rotates an active log file, it renames the current log file and then creates a new log file with the original name. The rotated log file is saved with the original file type and an appended timestamp.

The name of a rotated log file is in the form `<log_type>.timestamp`, where the components of the filename indicate the following:

- `<log_type>` specifies the log file type—system, error, or audit—that has been rotated.
- `timestamp` is a large integer that indicates the date and time the corresponding active log file was rotated. The date and time have the forms YYYYMMDD (Year, Month, Day) and HHmmSS (Hour, Minute, Second), in that order.

For example, an error log file rotated on July 28, 1998 at 12:36:24 would be named `error.19980728123624`.

Note The timestamp is expressed in standard Unix time: the number of seconds since midnight January 1, 1970.

Buffered Versus Unbuffered Logging

Certificate Management System supports buffered logging for all three types of logs—system, error, and audit. You can choose to configure the server for either buffered or unbuffered logging (see “Configuring Logs” on page 1058).

If you configure Certificate Management System for buffered logging, the server creates buffers for the corresponding logs, and it holds the messages in these buffers for as long as possible. The server flushes out the messages to the log files—which are maintained in your local file system—only when either of the following conditions occurs:

- The buffer gets full—the buffer gets full when the buffer size is equal to or greater than the value specified by the `bufferSize` configuration parameter. The default value for this parameter is 512 KB. (This parameter can be set from the CMS window; see "Configuring Logs" on page 1058.)
- The flush interval for the buffer is reached—the flush interval is reached when the time interval since the last buffer flush is equal to or greater than the value specified by the `flushInterval` configuration parameter. The default value for this parameter is 5 seconds. (This parameter can only be specified in the configuration file; it can't be set from the CMS window.)
- When current logs are read from CMS window—the server retrieves the latest log when it is queried for current logs.

If you configure the server for unbuffered logging, the server flushes out messages as they are generated to the log files. Because the server performs an I/O operation (writing to the log file) each time a message is generated, configuring the server for unbuffered logging decreases performance.

Rotation of Log Files

Certificate Management System supports automatic rotation of log files, which simplifies administration and facilitates backups. You are not required to manually retire the current log file and create a new one to hold subsequent logged events. You can back up all but the current log file in a directory at any time, without stopping the server or manually notifying the server to start a new log file. The parameters that control log rotation are specified in the configuration. To change the log file rotation parameters, see "Configuring Logs" on page 1058.

You should periodically archive or back up the rotated log files. For details, see "Archiving of Rotated Log Files" on page 1053.

Timing of Log File Rotation

Log files are rotated when either of the following conditions occur:

- The size limit for the corresponding file is reached—the size of the corresponding log file is equal to or greater than the value specified by the `maxFileSize` configuration parameter. The default value for this parameter is 100 KB. (This parameter can be set from the CMS window; see "Configuring Logs" on page 1058.)
- The age limit for the corresponding file is reached—the corresponding log file is equal to or older than the interval specified by the `rolloverInterval` configuration parameter. The default value for this parameter is 2592000 seconds (every hour). (This parameter can only be specified in the configuration file; it can't be set from the CMS window.)

Location of Rotated Log Files

Rotated log files are stored at the same location where the current or active log files are maintained. To find out where the active log files are located, see "Log File Locations" on page 1049.

Deletion of Log Files

Certificate Management System supports automatic deletion of rotated (or old) log files. The parameters that control log deletion are specified in the configuration file.

How to Conserve Disk Space

By default, Certificate Management System does not delete rotated log files automatically. Because the rotated log files are also saved in your local file system, these files eventually take up a considerable amount of disk space. You can avoid this problem by doing one of the following:

- Configure the server to automatically delete the rotated log files.

- Manually delete the log files from the local file system.

In either case, if you want to keep specific log files for future use, be sure to archive or back them up before they are deleted. For details, see "Archiving of Rotated Log Files" on page 1053.

Timing of Log File Deletion

If you configure Certificate Management System to delete rotated log files automatically, the server deletes these files when the life of the corresponding log file is equal to or older than the interval specified by the `expirationTime` configuration parameter. The default value for this parameter is 2592000 seconds (or every hour). Note that you can specify this parameter in the configuration file only; you can't set from the CMS window.

Archiving of Rotated Log Files

Log files, especially the audit log file, contain critical information. So it is good practice to periodically archive rotated log files to some archive media. Consider doing this whether you are manually deleting rotated log files or have configured the server to delete files automatically. You can archive log files by copying the entire `log` directory to your archive media.

Certificate Management System does not provide any tool or utility for archiving log files. Use the tools or utilities that your operating system provides for archiving.

Certificate Management System does, however, provide a command-line utility, called `signtool`, that allows you to sign log files before archiving them. This gives you a means of tamper detection. For details, see "Signing Log Files" on page 1075.

Chapter

26

Managing Logs

Each instance of Netscape Certificate Management System (CMS) maintains its own system, error, and audit log files. These files record events related to various CMS activities. By configuring logs, you can customize the contents in the log files.

This chapter explains how to use the CMS window to configure the system, error, and audit logs maintained by Certificate Management System, and how to monitor its activities by viewing log contents.

Before you attempt to configure or monitor logs, it's a good idea to read "Introduction to Logs" on page 1045.

The chapter has the following sections:

Management of Logs

You can manage CMS logs in two ways:

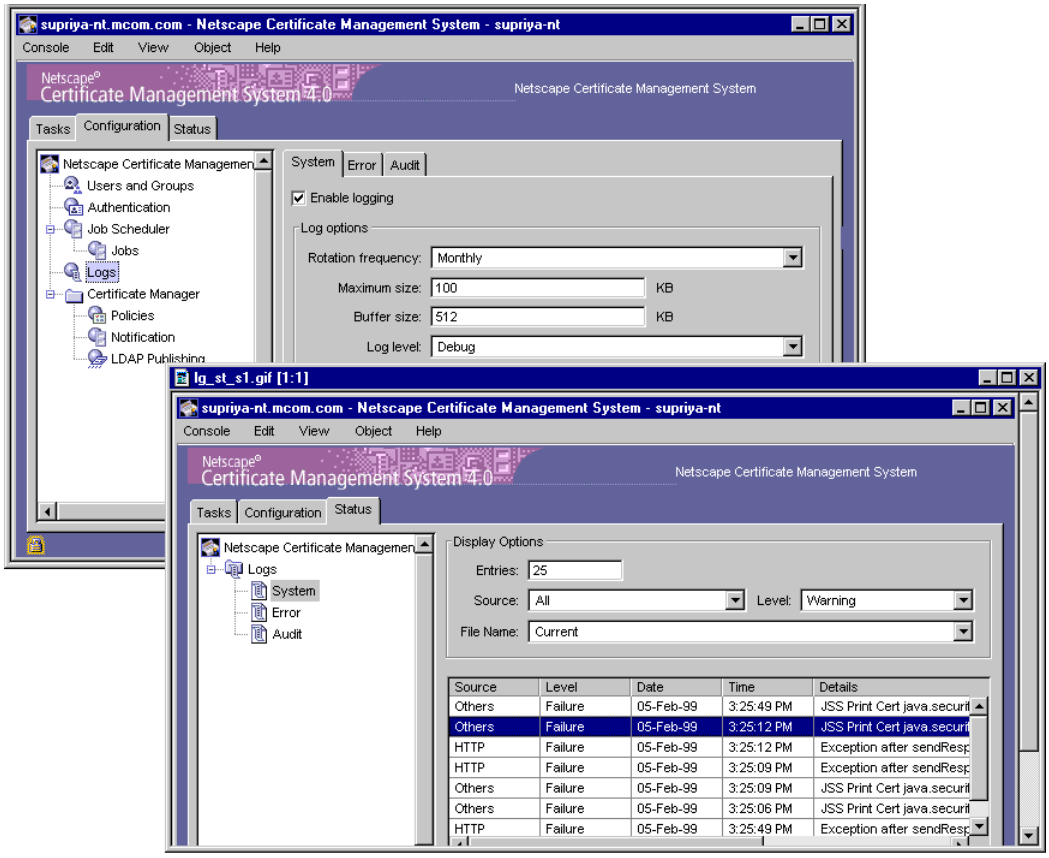
- By making changes to logging-specific configuration parameters from the CMS window and monitoring the resulting messages in the CMS window. See "Log Management From the CMS Window" on page 1056.
- By editing the configuration parameters in the configuration file. See "Log Parameters in the Configuration File" on page 1058.

The recommended method is to use the CMS window. However, for configuration parameters that are not shown in the CMS window, you may have to edit the configuration file.

Log Management From the CMS Window

The CMS window supports the configuration and monitoring of various CMS logs. In this window, you will find the `Logs` object in two places—in the navigation tree of the Configuration tab and in the navigation tree of the Status tab (see Figure 26.1).

Figure 26.1 Managing logs from the CMS window



The `Logs` object in the Configuration tab shows the current configuration of system, error, and audit logs and allows you to change it. For instructions on changing log configurations, see “Configuring Logs” on page 1058.

The `Logs` object in the Status tab shows messages logged by the server. For instructions on viewing logs, see “Monitoring Logs” on page 1064.

Log Parameters in the Configuration File

The sample configuration file on page 102 illustrates how information specific to logs appears in the configuration file, `CMS.cfg`. If you intend to change the configuration by editing the configuration file, be sure to follow the instructions provided in "Changing the Configuration by Editing the Configuration File" on page 86.

Configuring Logs

This section describes the procedures for configuring each type of CMS log:

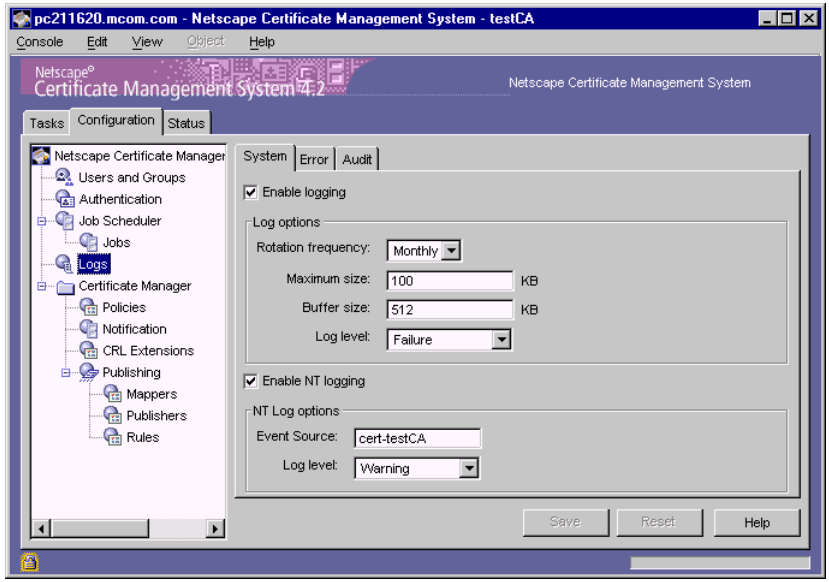
- Configuring System Logs
- Configuring Error Logs
- Configuring Audit Logs

Configuring System Logs

To configure the system log for a CMS instance:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. In the navigation tree, select Logs.

The System tab appears in the right pane. It shows the current configuration for the system log.



3. Check the "Enable logging" box if you want the server to log system-level messages to the appropriate CMS log file (see "Log File Locations" on page 1049). All the associated fields become available for you to enter information. Leave the box unchecked if you do not want the server to log messages of this type.

4. In the "Log options" section, specify information as appropriate:

   Rotation frequency. From the drop-down list, select the interval at which the server should rotate the active system log file. The available choices are Hourly, Daily, Weekly, Monthly, and Yearly. The default rotation interval is Monthly. For more information, see "Rotation of Log Files" on page 1051.

   Maximum size. Type the file size in kilobytes (KB) for the system log. The default file size is 100 KB. For more information, see "Rotation of Log Files" on page 1051.

   Buffer size. Type the buffer size in kilobytes (KB) for the system log. The default size for the buffer is 512 KB. For more information, see "Buffered Versus Unbuffered Logging" on page 1051.

Log level. From the drop-down list, select a log level. The choices are Debug, Info, Warning, Failure, Misconfiguration, Catastrophe, and Security. The default selection is Failure. For more information, see "Log Levels (Message Categories)" on page 1048.

5. This step is applicable to Windows NT system only.

   On a Windows NT system, check the "NT Event logging" box if you want the server to log system-level messages to the Event Log maintained by the system. All the associated fields become available for you to enter information. Leave the box unchecked if you do not want the server to log messages of this type to the Event Log.

   Event source. Specifies the CMS instance ID for which the system messages are to be logged. For example, the instance ID could be `cert-test CA`.

   Log level. From the drop-down list, select a log level. The choices are Debug, Info, Warning, Failure, Misconfiguration, Catastrophe, and Security. The default selection is Warning. For more information, see "Log Levels (Message Categories)" on page 1048.

6. To save your changes, click Save.

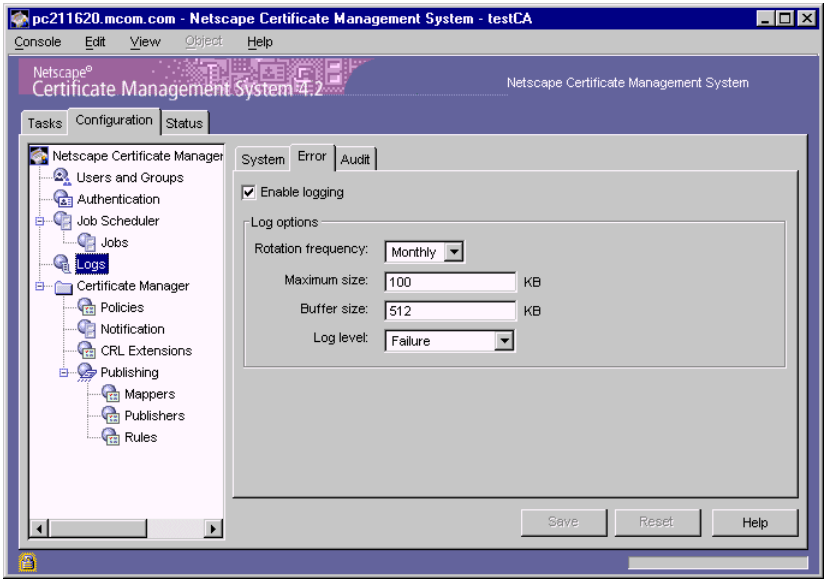
   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Configuring Error Logs

To configure the error log for a CMS instance:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. In the navigation tree, select Logs, and then in the right pane, select the Error tab.



3. If you want the server to log error messages, check the “Enable logging” box. All the associated fields become available for you to enter information. Leave the box unchecked if you do not want the server to log messages of this type.

4. In the “Log options” section, specify information as appropriate:

   Rotation frequency. From the drop-down list, select the interval at which the server should rotate the active error log file. The available choices are Hourly, Daily, Weekly, Monthly, and Yearly. The default selection is Monthly. For more information, see “Rotation of Log Files” on page 1051.

Maximum size. Type the file size in kilobytes (KB) for the error log. The default file size is 100 KB. For more information, see "Rotation of Log Files" on page 1051.

Buffer size. Type the buffer size in kilobytes (KB) for the error log. The default size for the buffer is 512 KB. For more information, see "Buffered Versus Unbuffered Logging" on page 1051.

Log level. From the drop-down list, select a log level. The choices are Debug, Info, Warning, Failure, Misconfiguration, Catastrophe, and Security. The default selection is Failure. For more information, see "Log Levels (Message Categories)" on page 1048.

5. To save your changes, click Save.

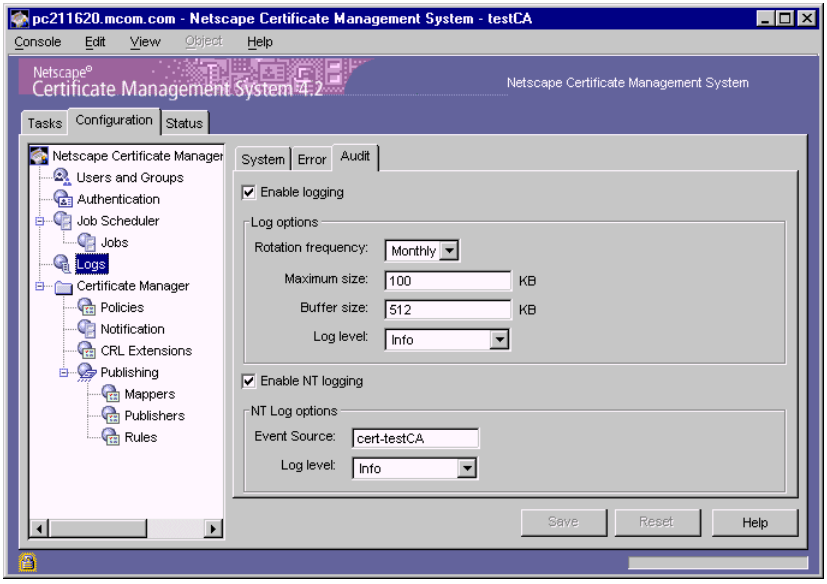
   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Configuring Audit Logs

To configure the audit log for a CMS instance:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).

2. In the navigation tree, click Logs, and then in the right pane, select the Audit tab.



3. If you want the server to log system-level messages, check the “Enable logging” box. All the associated fields become available for you to enter information. Leave the box unchecked if you do not want the server to log messages of this type.

4. In the “Log options” section, specify information as appropriate:

   Rotation frequency. From the drop-down list, select the interval at which the server should rotate the active audit log file. The available choices are Hourly, Daily, Weekly, Monthly, and Yearly. The default selection is Monthly. For more information, see “Rotation of Log Files” on page 1051.

Maximum size. Type the file size in kilobytes (KB) for the audit log. The default file size is 100 KB. For more information, see "Rotation of Log Files" on page 1051.

Buffer size. Type the buffer size in kilobytes (KB) for the audit log. The default size for the buffer is 512 KB. For more information, see "Buffered Versus Unbuffered Logging" on page 1051.

Log level. From the drop-down list, select a log level. The choices are Debug, Info, Warning, Failure, Misconfiguration, Catastrophe, and Security. The default selection is Info. For more information, see "Log Levels (Message Categories)" on page 1048.

5. This step is applicable to Windows NT system only.

   On a Windows NT system, check the "NT Event logging" box if you want the server to log audit messages to the Event Log maintained by the system. All the associated fields become available for you to enter information. Leave the box unchecked if you do not want the server to log messages of this type to the Event Log.

   Event source. Specifies the CMS instance ID for which the audit messages are to be logged. For example, the instance ID could be `cert-test CA`.

   Log level. From the drop-down list, select a log level. The choices are Debug, Info, Warning, Failure, Misconfiguration, Catastrophe, and Security. The default selection is Info. For more information, see "Log Levels (Message Categories)" on page 1048.

6. To save your changes, click Save.

   The CMS configuration is modified. If the changes you made require you to restart the server, you will be prompted accordingly. In that case, restart the server.

Monitoring Logs

When you have problems with Certificate Management System that require troubleshooting, you may find it helpful to check the error or informational messages that the server has logged. Also, by examining the log files you can monitor many aspects of the server's operation.

To facilitate this, the CMS window provides a simple mechanism for viewing the contents of both currently active and rotated audit, system, and error log files. The contents of the log file you choose to view are displayed in the form of a table. Each row is allocated to a specific log entry, with columns containing information such as the date and time the message was logged, the severity of the message, and a general description of the log. Once you open a log file for viewing, you can also do the following tasks:

- Read log file contents partially (by specifying the number of entries to be displayed)
- Filter log entries for specific services (by specifying the source)

This section covers the following topics on monitoring Certificate Management System by viewing log contents:

- Monitoring System Logs
- Monitoring Error Logs
- Monitoring Audit Logs
- Using System Tools for Monitoring the Server (Windows NT Only)

Monitoring System Logs

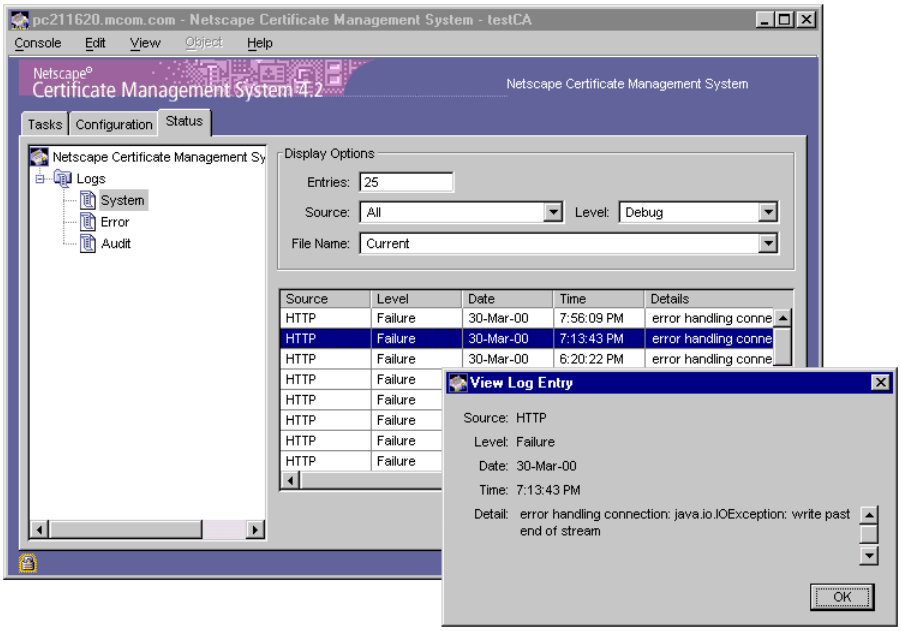
Certificate Management System maintains extensive system logs. These logs record various events and system errors for system monitoring and debugging. A system log records details such as the following:

- Each HTTP access invoked on the server
- Errors encountered, such as authentication failures, malformed universal resource indicators (URIs), invalid database password indications, and server start-up and shut-down messages
- Messages related to the status of certificate issuance or revocation, authentication failures for issuing-agent connections, and any errors related to the formatting of requests

You can view the contents of currently active as well as rotated system log files from the CMS window (see Figure 26.2).

If you have installed Certificate Management System on a Windows NT system, you can configure the server to log messages to Windows NT event log. For details, see “Logging to Windows NT Event Log”“Logging to Windows NT Event Log” on page 1072.

Figure 26.2 A sample active system log displayed in the CMS window



To view the contents of an active or rotated system log file:

1. Log in to the CMS window (see “Logging In to the CMS Window” on page 78).
2. Select the Status tab.
3. In the navigation tree, under Logs, select System.
4. In the Display Options section, specify your viewing preferences:

   Entries. Type the maximum number of entries to be displayed. When this limit is reached, Certificate Management System returns any entries it has located that match the search request. If you enter zero (`0`), no messages are returned. If you leave the field blank, the server returns every matching entry (no limit) regardless of the number found.

Source. Select the CMS component (or service) for which log messages are to be displayed. Depending on the components that write to this log file, the drop-down list shows one or more of the following: All, Registration Authority, Certificate Authority, Key Recovery Authority, HTTP, Internal Database, Authentication, Administration, LDAP, Request Queue, ACLs, User and Group, and Others. If you choose All, messages logged by all components that log to this file are displayed. For more information, see "Services That Are Logged" on page 1047.

Level. Select a message category that represents the log level for filtering messages. For more information on log levels, see "Log Levels (Message Categories)" on page 1048.

Filename. Select the log file you want to view. Choose Current to view the currently active system log file. For more information, see "Log File Naming Conventions" on page 1049.

5. Click Refresh.

   The table displays the system log entries. The entries are in reverse chronological order, with the most current entry placed at the top. Use the scroll arrows on the right edge of the panel to scroll through the log entries.

   For each entry you see the following details:

   Source. Indicates the CMS component or resource that logged the message.

   Level. Indicates the severity of the corresponding entry (explained Table 25.3 on page 1048).

   Date. Indicates the date on which the entry was logged.

   Time. Indicates the time at which the entry was logged.

   Details. Provides a brief description of the log.

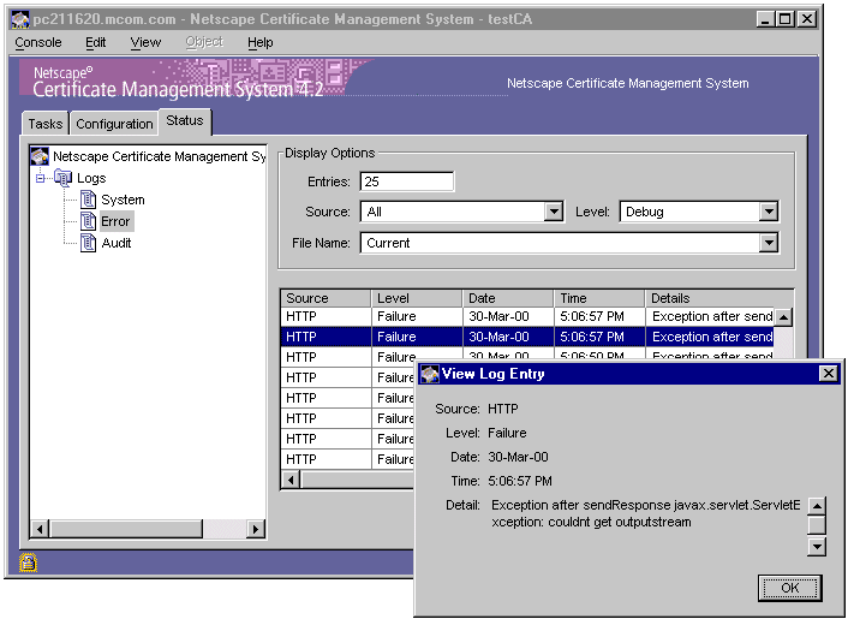
6. To view an entry in its entirety, either double-click it or select the entry and click View.

Monitoring Error Logs

The error log file contains errors the server has encountered since the log file was created; it also contains informational messages about the server, such as when the server was started. Incorrect user authentication is also recorded in the error log. Use the error log to find broken URL paths or missing files.

You can view the contents of currently active as well as rotated error log files from the CMS window (see Figure 26.3).

Figure 26.3 A sample active error log displayed in the CMS window



To view the contents of an active or rotated error log file:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Status tab.
3. In the navigation tree, under Logs, click Error.
4. In the Display Options section, specify your viewing preferences:

Entries. Type the maximum number of entries to be displayed. When this limit is reached, Certificate Management System returns any entries it has located that match the search request. If you enter zero (`0`), no messages are returned. If you leave the field blank, the server returns every matching entry (no limit) to the client regardless of the number found.

Source. Select the CMS component (or services) for which log messages are to be displayed. Depending on the components that write to this log file, the drop-down list shows one or more of the following: All, Registration Authority, Certificate Authority, Key Recovery Authority, HTTP, Internal Database, Authentication, Administration, LDAP, Request Queue, ACLs, User and Group, and Others. If you choose All, messages logged by all components that log to this file are displayed. For more information, see "Services That Are Logged" on page 1047.

Level. Select a message category that represents the level of logging to filter messages. For more information, see "Log Levels (Message Categories)" on page 1048.

Filename. Select the log file you want to view. Choose Current to view the currently active error log file. For more information, see "Log File Naming Conventions" on page 1049.

5. Click Refresh.

   The table displays the error log entries. The entries are in reverse chronological order, with the most current log placed at the top. Use the scroll arrows on the right edge of the panel to scroll through the log entries.

   For each entry you see the following details:

   Source. Indicates CMS component or resource that logged the message.

   Level. Indicates the severity of the corresponding entry (explained in Table 25.3 on page 1048).

   Date. Indicates the date on which the entry was logged.

   Time. Indicates the time at which the entry was logged.

   Details. Provides a brief description of the log.

6. To view an entry in its entirety, either double-click it or select the entry and click View.

Monitoring Audit Logs

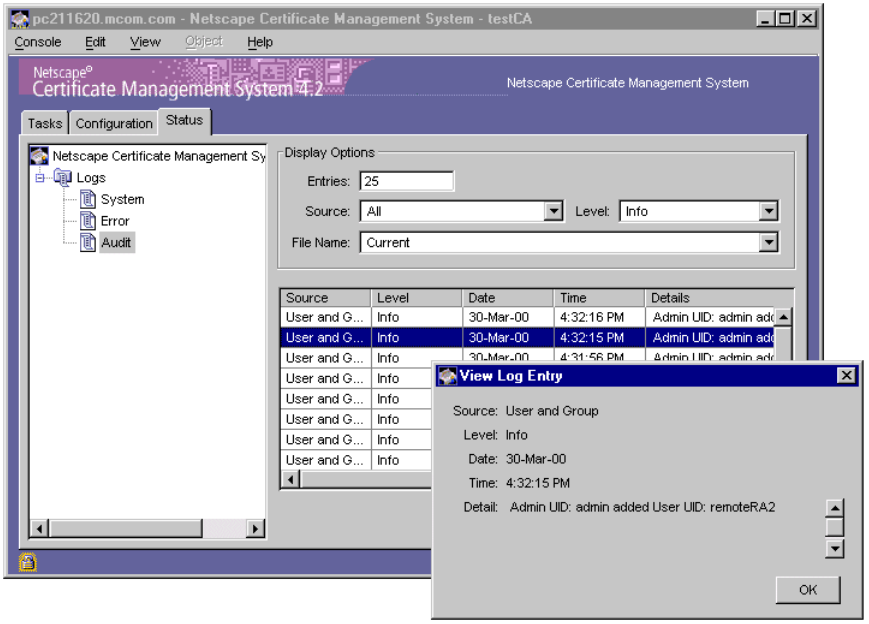
Certificate Management System maintains audit trails for all events—certificate requests, certificate renewal and revocation requests, CRL publication, and so on. These trails enable you to detect any unauthorized access or activity. The audit trails are logged and maintained in a file in your local file system.

If you have installed Certificate Management System on a Windows NT system, you can also configure the server to log audit messages to Windows NT event log. For details, see "Logging to Windows NT Event Log" on page 1072.

Important You should periodically examine and audit the CMS audit log for unusual activity. When examining the log, note in particular the log entries that fall under the Security-Related Events category (these are labeled `Security`).

You can view the contents of currently active as well as rotated audit log files from the CMS window (see Figure 26.4).

Figure 26.4 A sample active audit log displayed in the CMS window



To view the contents of an active or rotated audit log file:

1. Log in to the CMS window (see "Logging In to the CMS Window" on page 78).
2. Select the Status tab.
3. In the navigation tree, under Logs, select Audit.
4. In the Display Options section, specify your viewing preferences:

   Entries. Type the maximum number of entries to be displayed. When this limit is reached, Certificate Management System returns any entries it has located that match the search request. If you enter zero (`0`), no messages are returned. If you leave the field blank, the server returns every matching entry (no limit) regardless of the number it finds.

   Source. Select the CMS component (or resource) for which log messages are to be displayed. Depending on the components that write to this log file, the drop-down list shows one or more of the following: All, Registration Authority, Certificate Authority, Key Recovery Authority, HTTP, Internal Database, Authentication, Administration, LDAP, Request Queue, ACLs, User and Group, and Others. If you choose All, messages logged by all components that log to this file are displayed. For more information, see "Services That Are Logged" on page 1047.

   Level. Select a message category that represents the level of logging to filter messages. For more information, see "Log Levels (Message Categories)" on page 1048.

   Filename. Select the log file you want to view. Choose Current to view the currently active audit log file. For more information, see "Log File Naming Conventions" on page 1049.

5. Click Refresh.

   The table displays the audit log entries. The entries are in reverse chronological order, with the most current log placed at the top. Use the scroll arrows on the right edge of the panel to scroll through the log entries.

   For each entry you see the following details:

   Source. Indicates the CMS component or resource that wrote to the log file.

Level. Indicates the severity of the corresponding entry (explained in Table 25.3 on page 1048).

Date. Indicates the date on which this entry was logged.

Time. Indicates the time at which this entry was logged.

Details. Provides a brief description of the log.

6. To view an entry in its entirety, either double-click it or select the entry and then click View.

Using System Tools for Monitoring the Server (Windows NT Only)

If you have installed Certificate Management System on a Windows NT system, you can monitor the server with the system tools provided by Windows NT. This section explains how you can use the system tools.

- Logging to Windows NT Event Log
- Using Event Viewer
- Avoiding Event Log From Getting Filled

Logging to Windows NT Event Log

You can also configure Certificate Management System to write both audit and system logs to the event log of a Windows NT system. If you've installed Certificate Management System on a Windows NT system, the CMS window allows you to turn this feature on or off and to specify the levels for logging. For information on turning on or off to Windows NT Event Log, see "Configuring System Logs" on page 1058 and "Configuring Audit Logs" on page 1063.

Note that by default both the audit and system logs are enabled.

Using Event Viewer

In addition to logging messages to the log files maintained in your local file system, Certificate Management System can also log audit messages and system errors to the Windows NT Event log. To configure the server to log messages to Windows NT event log, see "Logging to Windows NT Event Log" on page 1072. If you configure the server to do so, you can use the system's tool called *Event Viewer* to monitor events related to your server.

More information about Event Viewer is available in your system documentation.

To monitor Certificate Management System by using Event Viewer:

1. In the Administrative Tools program group, double-click the Event Viewer icon.

2. From the Log menu, select Application.

   The Application log appears in Event Viewer. In this log, the *source* of any messages from Netscape Certificate Management System is the server's instance ID (if you didn't change the default values assigned to the `logNTAudit.NTEventSourceName` and `logNTSystem.NTEventSourceName` parameters).

3. From the View menu, choose Find to search for one of the Netscape labels in the log; use Refresh to see updated log entries.

4. Double-click a log entry to see additional information.

   The mapping between the CMS log levels and the Windows NT event type is shown in Table 26.1.

Table 26.1 Mapping between CMS log levels and Windows NT event log type

Windows NT log event type	CMS log level
Information	Debugging (0)
Information	Informational (1)
Warning	Warning (2)
Error	Failure (3)
Error	Misconfiguration (4)

Table 26.1 Mapping between CMS log levels and Windows NT event log type (Continued)

Windows NT log event type	CMS log level
Error	Catastrophic failure (5)
Error	Security-related events (6)

Avoiding Event Log From Getting Filled

When running Certificate Management System on a Windows NT system, if you don't configure the NT Event Log properly, the event log will get full. When this happens, you'll see an error message (see Figure 26.5) stating that the application log file is full.

Figure 26.5 Error message indicating event log is full

If you see this dialog box, you must clean up the application log immediately.

Here's what you should do:

1. From the Start menu on your desktop, select Programs, Administrative Tools (Common), and Event Viewer, in that order.

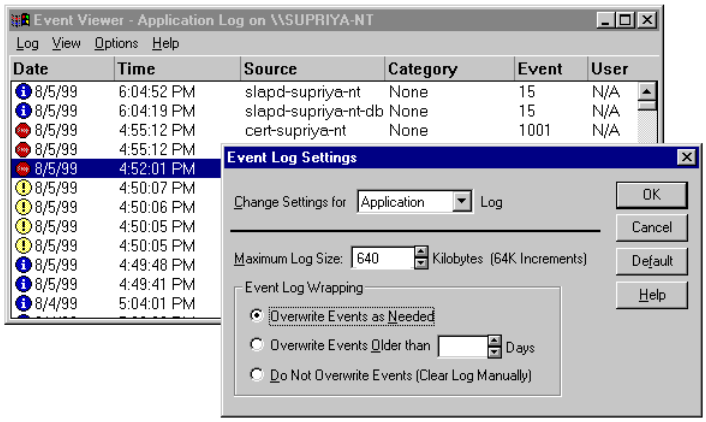
   This opens the Event Viewer window for the system.

2. From the Log menu, select Application.

   A checkmark to the left indicates it is selected.

3. From the Log menu, select Log Settings.

This opens the Event Log Settings window.



4. Enter the appropriate values:

   Change Settings for. Make sure that the Application log is selected in this box.

   Maximum Log Size. Select a reasonable size so that the event log doesn't get full in a short period of time.

   Event Log Wrapping. Select the "Overwrite Events as Needed" option.

5. Click OK.

6. Close the Event Viewer window.

Signing Log Files

Certificate Management System allows you to digitally sign log files before you archive them or distribute them for audit purposes. This feature enables you to check whether the log files have been tampered with since being signed.

For signing log files, you use a command-line utility called *Netscape Signing Tool*; for details about this utility, see Appendix F, "Netscape Signing Tool" in the online version of this document; to locate the online version of the

document, see "Where to Go for Related Information" on page 39. The utility uses information in the certificate (`cert7.db`), key (`key3.db`), and security module (`secmod.db`) databases of Certificate Management System.

Before you begin signing the log files, follow these guidelines:

- Determine the key pair you want to use for signing the log directory. Typically, you should use the Certificate Manager's (the CA's) signing key pair. Also find out the nickname of the certificate that corresponds to this key pair.
- If you have deployed many CAs, locate the CMS instance in which the CA you want to use is installed.
- Find out whether the key pair is in an internal or external token. If it is in an external token, make sure the token is currently installed. You will also need to know the password for the token.
- Determine which log files need to be signed. Put all the files that need to be signed in one or more directories. (The utility can sign a directory containing files; it cannot sign individual files.) Make sure these directories are in the host machine in which the CA is installed.
- Determine names for the output files; the output you receive will be a JAR file (which is a signed zip file). You may want to give names that will help you identify these JAR files easily in the future.

When you are ready with all this information, follow the procedure below to sign the log directories:

1. Go to the CMS instance in which the CA whose key pair you want to use for signing is installed.
2. Copy the security module database (`secmod.db` file) from the Administration Server configuration directory to the CMS configuration directory.

   The security module database is in this directory:

   `<server_root>/admin-serv/config`

   Copy it to this directory:

   `<server_root>/cert-<instance_id>/config`
3. Open a terminal window.

4. At the command prompt, run the following command with the appropriate information:

   ```
   signtool -d <secdb_dir> -k <cert_nickname> -Z <output>
   <input>
   ```

   `<secdb_dir>` specifies the path to the directory that contains the certificate, key, and security module databases for the CA. This must be the same path you used to copy the security module database in step 2.

   `<cert_nickname>` specifies the nickname of the certificate you want the utility to use for signing.

   `<output>` specifies the name of the JAR file (a signed zip file).

   `<input>` specifies the path to the directory that contains the log files.

   For example, in a Windows NT system, your command might look like this:

   ```
   signtool -d c:\netscape\server4\cert-testCA\config -k
   testCAsigningcertificate -Z log_err_02_99.jar
   c:\archive\logs
   ```

   where `c:\netscape\server4\cert-testCA\config` is the path to the certificate, key, and security module databases (`secdb_dir`).

   `testCAsigningcertificate` is the certificate nickname (`cert_nickname`).

   `log_err_02_99.jar` is the name of the JAR file (output).

   (`input`) is `c:\archive\logs` is the directory to be signed.

10

Issuance and Management of End-Entity Certificates

Chapter

27

Issuing and Managing End-Entity Certificates

This chapter explains how Netscape Certificate Management System (CMS) issues and manages end-entity certificates.

The chapter has the following sections:

Certificate Issuance to Servers

For Certificate Management System to generate a server certificate, it must receive the certificate signing request (CSR) from the server that needs the certificate. This request must be initiated by the administrator of the specific server requiring the certificate.

SSL-enabled servers (or servers that are capable of using certificates for security) provide mechanisms for generating a CSR based on new or existing key pairs. For example, servers that belong to the Netscape's version 4.x server family come with a wizard that walks an administrator through the entire

process of requesting a server certificate and installing it in the server's certificate database. For information on this wizard, see "Obtaining and Installing a Certificate" in *Managing Servers with Netscape Console*.

Once an administrator generates a CSR for a server, he or she must paste it into the appropriate server enrollment form hosted by a Registration Manager or Certificate Manager, and then submit the request. Upon receipt of the request, Certificate Management System responds as follows:

1. Verifies the validity and authenticity of the request.

   The authentication mechanism that Certificate Management System uses is based on the authentication mechanism specified in the enrollment form the administrator uses to submit the certificate request. For example, if the enrollment form was configured to employ directory-based authentication, Certificate Management System checks the configured directory for the appropriate information. On the other hand, if the enrollment form specifies manual authentication, the request gets queued and awaits approval by an agent.

2. Subjects the request to policy checks.

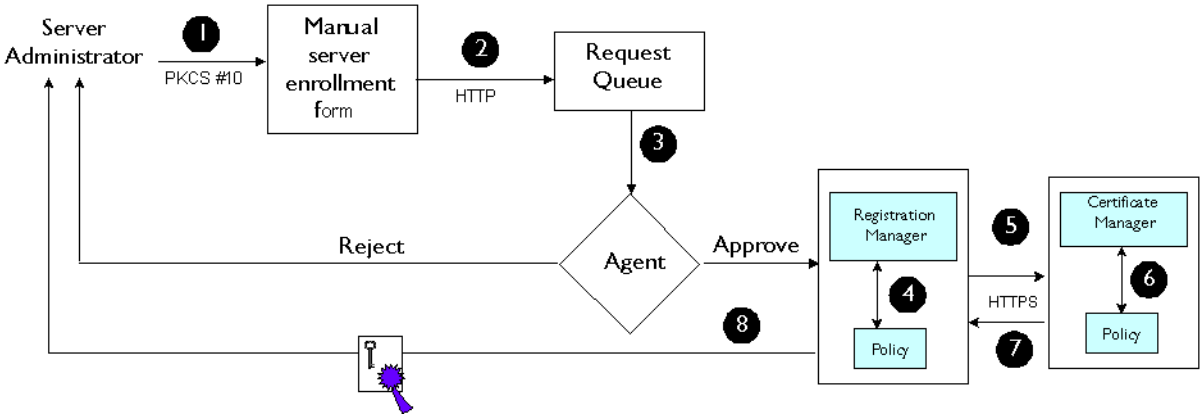
   If the request passes all the policy rules, Certificate Management System generates the server certificate and sends it to the email address specified in the server certificate request (the enrollment form includes a field for the administrator to enter this information). Otherwise, Certificate Management System logs an error message.

Upon receipt of the certificate, the server administrator installs the certificate in the server's certificate database.

How the Manual Server Enrollment Process Works

Figure 27.1 illustrates how Certificate Management System issues a server certificate in a deployment scenario involving a Registration Manager acting as an enrollment authority to a Certificate Manager. The server certificate is requested via a manual enrollment form hosted by the Registration Manager.

Figure 27.1 Server (or site) certificate issuance



These are the steps shown in Figure 27.1:

1. The server administrator goes to the manual enrollment form hosted by the Registration Manager, pastes in the certificate signing request in PKCS #10 format, completes the other information in the enrollment form, and submits the form.

   (If the enrollment port is HTTPS, the administrator should visit the link that delivers the CA's certificate chain and download the chain into the browser that he or she will use for server enrollment.)

2. The Registration Manager verifies the authenticity of the request. Because the request requires manual authentication, the Registration Manager stores the request in the queue for agent approval.

3. An agent processes the request and either rejects or approves it.

4. The Registration Manager picks up the approved request and subjects it to policy checks.

5. If the request passes the Registration Manager's policy checking, the Registration Manager submits the request to the Certificate Manager for signing. The Certificate Manager verifies the authenticity of the Registration Manager by verifying the certificate presented by it. If it is a trusted Registration Manager, the Certificate Manager accepts the request.

6. The Certificate Manager subjects the request to its own policy checks.

7. If the request passes Certificate Manager's policy, it signs the request immediately and returns the certificate to the Registration Manager. The Registration Manager then delivers the certificate to the administrator. Optionally, the Certificate Manager may publish the certificate to the corporate directory.

   If the Certificate Manager's policy requires additional information, the administrator will be directed to return later to pick up the certificate. The administrator may need to query the Registration Manager using the certificate request number to see whether the certificate has been issued. Alternatively, the Registration Manager can be configured to email the user when the certificate is ready for pick up. See "Notifications of Certificate Issuance to End Entities" on page 450.

8. The Registration Manager delivers the server SSL certificate to the email address specified in the enrollment form. Optionally, the Registration Manager may publish the certificate to the corporate directory.

Getting Server SSL Certificates for Netscape Servers

To enable a server to establish SSL connections, you need to get a certificate that identifies the server. You can get a certificate for a server by submitting a request to Certificate Management System.

To generate the actual request, you (or the server administrator) need to use the server that requires the certificate. This is required because the private key must be stored with the server that will use it.

The following section explains how to request a server SSL certificate for Netscape servers. The instructions apply mainly to requests from servers other than CMS subsystem server—for example, Netscape Enterprise, Administration, and Directory Servers. To request a certificate for a CMS subsystem, follow the instructions in "Getting New Certificates for the Subsystems" on page 277.

Getting Certificates for Version 3.x Servers

To get a certificate for a server in the Netscape version 3.x server family (for example, Netscape Administration Server 3.x) follow the procedure below:

- Step 1. Generate a Server Certificate Request for Your Server
- Step 2. Submit the Server Certificate Request to Certificate Management System
- Step 3. Install Your Server's SSL Certificate
- Step 4. Accept a CA as Trusted in Your Server
- Step 5. Verify Your Server's SSL and CA Certificates

Step 1. Generate a Server Certificate Request for Your Server

To generate the certificate signing request for a server:

1. Open a web browser window.
2. Go to the Administration Server, and use the Server Selector to access the Server Manager for your server.
3. Follow the directions presented there to generate a new key pair which you will then get certified (you will use this key pair to generate a certificate signing request).

   Alternatively, you can use any other tool provided with your server to generate the key pair; see the documentation for your server.
4. Once you have generated a key pair, follow the directions presented to generate a certificate signing request (CSR).
5. In the Certificate Authority field, enter your own email address.

   The server mails the request to the address specified in this field.
6. Submit the form.

   The server generates and displays a CSR.
7. Copy the CSR, including the `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----` marker lines, to a text file. For example:

   ```
   -----BEGIN NEW CERTIFICATE REQUEST-----
   ```

```
MIIBBzCBsgIBADBPMQswCQYDVQQGEwJVUzEoMCYGA1UEChMfTmV0c2NhcGUgRGlyZWN0b3J5IFB1Y
mxpY2F0aW9uczEWMBQGA1UEAxMNZHVtcC5tY29tLmNvbTBaMA0GCSqGSIb3DQEBAQU2nfjiMEYCQQ
CksMRaLGdfp4m0OiGcgijG5KgOsyRNvwGYW7kfW+8mmijDtZRjYNjjcgpF3VnlsbxbclX9LVjjNLC
57u37XZdAgEDoAAwDQYJKoZIhvcNAQEEBQADQQCYUTnUtCVGyNrYGSfydclqiovxy1fRD1z23zg+e
BPK7n85UyE4r5zGZjDsMYr172ytfAFL7DeG83DWzr8Z
-----END NEW CERTIFICATE REQUEST-----
```

Next, you need to paste this request into the server enrollment form hosted by Certificate Management System.

Step 2. Submit the Server Certificate Request to Certificate Management System

To submit the server certificate request to Certificate Management System:

1. Open a web browser.
2. Go to the server enrollment form (the page that allows you to submit a server certificate request).

   By default, the enrollment forms are at this location:

   `https://<host_name>:<end_entity_HTTPS_port>` or

   `http://<host_name>:<end_entity_HTTP_port>`
3. In the Enrollment tab, unser Server, select SSL Server.

   The form for requesting SSL server certificate appears.
4. Complete the request form with the information that Certificate Management System needs to create a certificate for your server.

   In general, you will be required to enter the following information:

   - In the certificate request text area, paste the CSR that you copied to the text file, including the `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----` marker lines.
   - In the contact information section, enter values to identify yourself. These values will be used by the CA, if the need arises. For example, if there are any questions or problems with the certificate request, the CA administrator or agent will use this information to contact you. Also, be sure to enter your email address. This is the address where the CA will send the certificate once it has been issued.

- In the additional comments section, enter any additional information that might help the issuing agent process the request. For example, you might want to enter the name of the person who instructed you to obtain a certificate or some other administrative information.

5. Submit the request.

You should receive notification from Certificate Management System or an issuing agent (depending on which enrollment form you used) when your request is processed. The notification will contain your certificate, along with information on how to install the new certificate into your server. The notification may also mention that you need to install the CA's certificate as a trusted CA. Check the notification message for details.

Step 3. Install Your Server's SSL Certificate

To install the server SSL certificate on your server:

1. Open a web browser window.

2. Go to the Administration Server, and use the Server Selector to access the Server Manager for your server.

3. Follow the directions presented there to install the certificate.

   In general, you will be required to specify or enter the following information:

   - Whether the certificate is for this server. Be sure to select the option that says the certificate is for this server.

   - A name (or nickname) for the certificate. This name will be displayed in the list of certificates installed on this server.

   - The certificate, in base-64 encoded format. Open the email sent to you by the CA, locate and copy the portion that begins with `-----BEGIN CERTIFICATE-----` and ends with `-----END CERTIFICATE-----`, and paste it into the text area in the form.

   - The encryption alias. Enter the alias for your server.

4. Follow the prompts and add the certificate to your server's certificate database.

5. Stop and restart Administration Server for the changes to take effect.

   The server decrypts the message, extracts the certificate, and saves it to the directory you specified.

Step 4. Accept a CA as Trusted in Your Server

In both Netscape clients and servers, CAs can be either *trusted* or *untrusted*. If a CA is trusted, Netscape clients and servers accept the certificates that have been issued by that CA. For the server to accept (during SSL client authentication) client certificates that have been issued by Certificate Management System, you must import its certificate chain into the certificate database of your server.

To view this chain in a format that can be used by Netscape servers:

1. Go to the home page of Certificate Management System.

   By default, the home page is at this location:

   `https://<host_name>:<end_entity_HTTPS_port>`

2. Click Accept "This Authority in Your Server."

3. Specify how you want Certificate Management System to display the certificate chain.

   You can choose to display the entire certificate chain (in a single block) or individual certificates in the chain. The entire certificate chain is in PKCS #7 format. If you are using an older server that does not recognize the complete certificate chain format, you may need to display each individual certificate in the chain (for example, a version earlier than Netscape server 2.0 releases).

4. Specify how you want to trust this CA.

   You can choose to trust only the CA you are accessing or all authorities whose certificates are included in the chain.

5. Click Present Certificate Chain.

   If you chose to display the whole chain for importing into your server, the certificate chain is displayed in a format similar to this:

```
-----BEGIN CERTIFICATE-----

MIIBtgYJYIZIAYb4QgIFoIIBpzCCAZ8wggGbMIIBRaADAgEAAgEBMA0GCSqGSIb3DQEBBAUAMFcxC
zAJBgNVBAYTAlVTMSwwKgYDVQQKEyNOZXRzY2FwZSBDb21tdW5pY2F0aW9ucyBDb3Jwb3JhdGlvbj
EaMBgGA1UECxMRSXNzdWluZyBBdXRob3JpdHkwHhcNOTYxMTA4MDkwNzM0WhcNOTgxMTA4MDkwNzM
```

```
0WjBXMQswCQYDVQQGEwJVUzEsMCoGA1UEChMjTmV0c2NhcGUgQ29tbXVuaWNhdGlvbnMgQ29ycG9y
YXRpb24xGjAYBgNVBAsTEUlzc3VpbmcgQXV0aG9yaXR5MFowDQYJKoZIhvcNAQEBBQADSQAwRgJBA
OBiQPcK8851jjQXA2GBsaKNFg6pYaM3qhQhM0w5EIy6P1ttMjc5MlPIzZHdlgNdQLzaNoLMVKjOV5
sBp+ffkCAQMwDQYJKoZIhvcNAQEEBQADQQCWPU4gI5uaWM3EAbXfhQ
-----END CERTIFICATE-----
```

6. Open a new web browser window.

7. Go to the Administration Server, and use the Server Selector to access the Server Manager for your server.

8. Follow the directions presented there to install the certificate chain.

   In general, you will be required to specify or enter the following information:

   - Whether the certificate is for this server or a trusted CA. Be sure to select the option that says the certificate is for a trusted certificate authority (CA).
   - A name (or nickname) for the certificate chain. This name will be displayed in the list of certificates installed on this server.
   - The certificate chain, in PKCS #7 format. In the original browser window (the window displaying the encoded certificate chain), copy the portion that begins with `-----BEGIN CERTIFICATE-----` and ends with `-----END CERTIFICATE-----`, and paste it into the message text area in the form.

9. Save your changes.

10. Stop and restart your Administration Server.

Step 5. Verify Your Server's SSL and CA Certificates

Before activating your server for SSL connections, you can verify whether you have installed your server's SSL and CA certificates correctly.

1. Open a web browser window.

2. Go to the Administration Server, and use the Server Selector to access the Server Manager for your server.

3. Follow the directions there to get to the area that allows you to manage your server's certificates.

4. Scroll to the bottom of the list to find the SSL and CA certificate chain you installed (identified by the nicknames you specified).

   If you find both of them, your server is ready for SSL configuration. If not, you must go through the steps again to correctly install whichever certificate is missing.

Getting Certificates for Netscape Version 4.x Servers

For Netscape version 4.x servers, you can use the Certificate Setup Wizard provided by Netscape Console to get new certificates, renew existing certificates, and install certificates in the database of a server. For information about this wizard, see *Managing Servers with Netscape Console*. To locate an online version of this book, open the `<server_root>/manual/index.html` file.

Note that there are two ways in which you can submit the certificate signing request to Certificate Management System:

- Submit the request (which is in the form of a base-64 encoded blob) directly from the wizard; in this method, you need not copy the request to a text file.

- Submit the request manually by pasting the request (which is in the form of a base-64 encoded blob) in to the Certificate Manager's server enrollment form; in this method, you need to copy the request when the wizard displays it.

To submit the request to the Certificate Manager directly from the wizard:

When the wizard generates the certificate signing request for the key size and type you specified, you're presented with the opportunity to choose how you want to submit the request to the CA. The choices include the following:

To CA's email address. This option allows you to send the CSR to the CA administrator's email address. The administrator will then be required to submit the request to the CA by pasting the CSR in the CA's server enrollment form.

To CA's URL. This option allows you to submit the CSR to the CA directly. To submit the CSR to the Certificate Manager, you should enter, depending on the end-entity port you want to use, either of the following URL:

`http://<CA's_host_name>:<end_entity_port>/enrollment` or
`https://<CA's_host_name>:<end_entity_SSL_port>/enrollment`

Note that the request submitted to the CA's URL gets queued for approval by the Certificate Manager agent.

To submit the server certificate request to Certificate Management System manually:

1. Open a web browser window.
2. Go to the End Entity Services interface of the Certificate Manager (or a Registration Manager that's connected to the Certificate Manager) by entering either of these URLs:

   `https://<host_name>:<end_entity_HTTPS_port>` or

   `http://<host_name>:<end_entity_HTTP_port>`
3. In the left frame, under Server, select SSL Server.
4. In the server-enrollment form that appears, enter the required information:

   PKCS#10 Request. Paste the base-64 encoded blob, including the `-----BEGIN NEW CERTIFICATE REQUEST-----` and `-----END NEW CERTIFICATE REQUEST-----` marker lines, you copied to the text file earlier.

   Name. Type your name.

   Email. Type your business email address, for example, `jdoe@someCompany.com`.

   Phone. Type your business phone number.

   Additional Comments. Type any information that will help you identify this request in the future or will help the person who will process this request.
5. Click Submit.

CEP Enrollment

Cisco routers support the use of certificates for authentication, encryption, and tamper detection by using the IP Security (IPSec) protocol. Certificate Management System supports Cisco's PKI protocol, the Certificate Enrollment Protocol (CEP); this protocol runs over HTTP and provides its own form of encryption. For an overview of certificate authority support for IPSec, see the information available at this URL:

```
http://www.cisco.com/warp/public/cc/cisco/mkt/security/encryp/
prodlit/821_pp.htm
```

You can issue certificates to routers and CEP-compliant Virtual Private Network (VPN) clients using Certificate Management System. Routers use certificates to authenticate each other and to establish an encrypted IPSec channel between them; all TCP/IP communication passes through this encrypted channel.

Note that Certificate Management System by default supports issuance of certificates to routers and VPN clients using the CEP-based enrollment. However, publishing of these certificates to an LDAP-compliant directory is not turned on by default because routers and VPN clients need to have access to an LDAP directory in order to fully support various functions, such as certificate and CRL retrieval. This section explains how to set up a Certificate Manager to issue certificates to routers and CEP-compliant Virtual Private Network (VPN) clients. The section also describes how to configure the Certificate Manager to publish these certificates and certificate revocation lists (CRLs) to an LDAP-compliant directory.

You may configure the Certificate Manager to publish to any LDAP-compliant directory, but if you do not have one available, you can use the one supplied with Certificate Management System. Certificate Management System comes with Netscape Directory Server, which is an LDAP-compliant directory. When you install Certificate Management System, two instances of Netscape Directory Server are automatically created in the same server group in which Certificate Management System is installed—one of the Directory Server instances is identified as the *configuration directory* and the other *internal database*. For publishing certificates and CRLs you may use the configuration directory, but not the internal database. The internal database is configured for exclusive use by Certificate Management System; see "Internal Database" on page 163.

There are two ways to set up CEP enrollment:

- CEP Enrollment Using the Script
- Setting up CEP Enrollment Manually

The recommended is to use the interactive script.

CEP Enrollment Using the Script

Certificate Management System provides a menu-driven, interactive script to automate the CEP enrollment process. To invoke the script:

1. Go to the Certificate Manager's host system.
2. Open a command-line window.
3. Go to this directory: `<server_root>`
4. Enter the following at the prompt:

   ```
   % install/perl/bin/cert/tools/cepconfig.pl
   ```

   The main menu shows up.

```
                              CEPCONFIG

This script can be used to configure any instance of CMS.
Configuration tasks include the following:

- Adding/removing CEP services
   • You can configure different services, responding to
     different URLs in CMS. This enables different
     authentication and policy options to be set for different
     types of client (e.g. router and VPN)

- Enabling/disabling LDAP publishing of certificates and CRLs
   • Quick-and easy set up the publishing directory.
   • Turns on publishing.
   • Adds CRL distribution point to certificates.
_____________________________________________________________
Press ENTER to continue....
```

5. Follow the on-screen instructions to set up CEP enrollment.

Setting up CEP Enrollment Manually

The information covered in this section explains how to set up CEP enrollment manually. Note that the instructions are written with these assumptions:

- That you will publish certificates and CRLs to the configuration directory. For more information about the configuration directory, see *Managing Servers with Netscape Console*. To locate this document, open this file: `<server_root>/manual/index.html`
- That you will publish certificates and CRLs to the same tree in the configuration directory; you may customize this if you desire. We recommend that you publish to a tree named after the `O` attribute in your CA signing certificate. Router certificates will also need to have an `O` inserted in the subject name; this can be done automatically. This section refers to the name of this tree as `Base DN`.

If you want to publish to any other LDAP-compliant directory, read Part 7 “Publishing” on page 713.

To set up CEP enrollment manually, follow these steps:

- Step 1. Set up the Directory for Publishing Certificates and CRLs
- Step 2. Configure the Certificate Manager for Publishing Certificates and CRLs
- Step 3. Set up Automated Enrollment (optional)
- Step 4. Set Up Multiple CEP Services (optional)

Step 1. Set up the Directory for Publishing Certificates and CRLs

The section “Step 2. Set Up the Directory for Publishing” on page 789 contains information on setting up Netscape Directory Server for publishing certificates and CRLs—it covers directory schema required for publishing certificates and the attributes to which a Certificate Manager publishes end-entity certificates and CRLs.

For the configuration directory to support publishing of certificates and CRLs, you need to verify two things:

- The Directory Server schema—verify that the directory schema can accommodate router and VPN client certificates. You may need to update the Directory Server's schema. The reason for this is, if you plan on publishing certificates from routers, they may need to be published with the same DN as their certificate subject names. For example, if the certificate subject name contains `UnstructuredAddress` or `UnstructuredName` components, you may need to add them to the directory schema.

  ```
  unstructuredAddress, 1.2.840.113549.1.9.7, string
  unstructuredName, 1.2.840.113549.1.9.8, string
  ```

  To modify the schema you can use the Directory Server window, which can be launched from within Netscape Console. Alternatively, you can prepare an LDIF file with the changes you want to make and then run the LDAP *modify* command. Check the directory documentation for instructions.

- The Directory Server port—note the port number assigned to the configuration directory; it must be 389. If you installed Certificate Management System with the default choices, you may skip this step; the default port assigned to the configuration directory is 389. To find out the port number assigned to Directory Server, check it's configuration file (which is at `<server_root>/slapd-*/slapd.oc.conf`). Alternatively, you can also find and change the port number from Netscape Console.

Step 2. Configure the Certificate Manager for Publishing Certificates and CRLs

In this step, you should configure the Certificate Manager to issue router and VPN-client certificates with *CRL Distribution Point Extension* and to publish the certificates to a directory.

- Create an instance of the mapper plug-in named `LdapExactMapper` and of the publisher plug-in named `LdapUserCertPublisher`. Once you create these instances, you should create a publishing rule for publishing router certificates. For instructions, see "Step B. Add Mappers, Publishers, and Publishing Rules" on page 816.

  Note that the publishing rule must be configured to use the mapper and publisher you create for router certificates. In addition, the predicate expression must be set to `HTTP_PARAMS.certType==CEP-Request`.

- Configure CRL publishing details; for instructions, see “Step 4. Configure the Certificate Manager to Publish CRLs” on page 823.
- Identify the directory for publishing. For instructions, see “Step 5. Identify the Publishing Directory” on page 830.
- Create an instance of the policy plug-in named `CRLDistributionPointsExt`, which is explained in “CRL Distribution Points Extension Policy” on page 591, for router certificates. This extension, if present in a certificate, enables the user of the certificate to find revocation information pertaining to that certificate. When you create an instance of the `CRLDistributionPointsExt` plugin, be sure to leave the `issuerName` and `issuerType` fields blank and to enter `HTTP_PARAMS.certType==CEP-Request` in the `predicate` field.
- Stop the Certificate Manager and edit the configuration file to include the following lines:

  ```
  eeGateway.cep.cep1.appendDN=O=<BASE DN>
  eeGateway.cep.cep1.createEntry=true
  eeGateway.cep.cep1.entryObjectClass=cep
  eeGateway.cep.cep1.url=/cgi-bin/pkiclient.exe
  ```

  A description for each of the above parameters are provided in Table 27.1.

Table 27.1 CEP service-related parameters in the configuration file

Parameter	Description
`appendDN`	Specifies the DN component appended to the DN the router requests. You must have a constant component in the DN which exists in the certificate to be able to publish.
`createEntry`	Specifies whether to create an entry in the directory before publishing the certificate. Note that to publish a certificate, an entry must already exist for the DN in the directory. • Enter `true` if you want the Certificate Manager to create an entry if one does not already exist (`true`/`false`). • Enter `false` if an entry already exists in the directory and you don’t want the server to create one.

Table 27.1 CEP service-related parameters in the configuration file (Continued)

Parameter	Description

`entryObjectClass`	Specifies the type of object to assign to the new entry. By default, this `cep`, and should not be changed. Note that when `createEntry=true`, the Certificate Manager will attempt to create an entry for the user. The directory hierarchy must be set up correctly beforehand to accept new entries. To clarify this by an example, if you expect the Certificate Manager to be able to create an entry for a certificate with DN `CN=John Doe, OU=Accounting, O=Company, C=US`, you must have already created three (3) directory entries for
	
	`C=US`
	`O=Company, C=US`
	`OU=Accounting, O=Company, C=US`
	
	You can do this with the help of the `ldapmodify` command and an LDIF file with the following information:
	
	`dn: C=US`
	`changetype: add`
	`objectclass: top`
	`objectclass: country`
	`c: US`
	
	`dn: O=Company, C=US`
	`changetype: add`
	`objectclass: top`
	`objectclass: organization`
	`o: Company`
	
	`dn: OU=Accounting, O=Company, C=US`
	`changetype: add`
	`objectclass: top`
	`objectclass: organizationalunit`
	`ou: Accounting`
	
	`dn: cn=config,cn=ldbm`
	`changetype: modify`
	`add: nsslapd-suffix`
	`nsslapd-suffix: C=US`
	
	Also, note that in this case, `C=US` is the root of your directory, because it is the sole component of the DN. To create a new root, you must first add a new suffix, which is done by adding this to your LDIF file (see the last section of the LDIF file above). For more details, check *Netscape Directory Server Administrator's Guide*.

Table 27.1 CEP service-related parameters in the configuration file (Continued)

Parameter	Description
`url`	Specifies the URL for CEP enrollment. It is used if the router requests a subject name such as `unstructuredAddress=1.2.3.4+unstructuredName=fred.siroe.com`. You will need to append the DN to add-on `O=siroe.com` as otherwise publishing to the directory will not work.

Once this is done, you may configure the Certificate Manager for automated enrollment. To do so, follow the steps in the next section. If you do not want to configure the server for automated enrollment, restart the server.

Step 3. Set up Automated Enrollment

As a part of enrolling for a certificate (via CEP), a router administrator or VPN-client user needs to start the enrollment process, which in turn asks the user for information such as the following:

- The CA's identity
- The CEP enrollment URL
- A challenge password
- The serial number and IP address

Some of the information a user enters, such as the serial number and IP address, goes in to the subject name in the CEP request. Information such as the CA's identity and enrollment URL enables the router to connect to the valid CA to make the certificate request. The challenge password, if specified, enables the user to authenticate to the server during enrollment and to revoke the certificate, if needed, by presenting the same password again. (See "Certificate Issuance to Routers or VPN Clients" on page 1105.)

You can configure the Certificate Manager to use either the challenge password or the subject name (all or a part of it) as an authentication token during a CEP enrollment, thus enabling users to get router certificates without any action on the part of the Certificate Manager agent.

To aid you in implementing the automated CEP enrollment process, Certificate Management System comes with an authentication plug-in module named `FlatFileAuth`. This plug-in is available in source-code form in the CMS samples package in this directory:

```
<server_root>/cms_sdk/samples/authentication
```

In order for the Certificate Manager to recognize the `FlatFileAuth` plug-in and use it for authenticating CEP-based certificate requests, you must do the following:

- Register the plugin in the CMS authentication framework; for instructions, see "Registering an Authentication Module".
- Create an instance of the plug-in; for instructions, see "Step 4: Add an Authentication Instance" on page 395.

You can do this either via the CMS window or by adding the required parameters to the Certificate Manager's configuration file (`CMS.cfg`). The configuration parameters of the `FlatFileAuth` plug-in are as follows:

```
eeGateway.cep.cep1.authName=flatfile
auths.instance.flatfile.fileName=<full_pathname_of_password_
file>
auths.instance.flatfile.authAttributes=pwd
auths.instance.flatfile.keyAttributes=UNSTRUCTUREDNAME
auths.instance.flatfile.pluginName=flatfilePlugin
auths.instance.flatfile.deferOnFailure=false

auths.impl.flatfilePlugin.class=com.netscape.certsrv.authent
ication.FlatFileAuth
```

A description for each of the above listed parameters are provided in Table 27.2.

Table 27.2 Configuration parameters defined in the FlatFileAuth plug-in

Configuration parameter	Description
`authName`	Provides a reference to the auths.instance authentication plug-in described in the `auths.instance.*` configuration parameters. If you want to turn off automated enrollment for CEP-based requests, delete this parameter from the configuration file.
`fileName`	Specifies the filename of an authentication-token file. You prepare this file as a part of setting up an automated CEP enrollment as explained in Step 4-B. Be sure to use the full path name.
`keyAttributes`	Specifies a comma-separated list of attributes in the request which together, uniquely identify an entry in the authentication-token file. Note that these attributes must be present in the request and in the password file for the authentication to succeed.
`authAttributes`	Specifies a comma-separated list of attributes from the CEP request which must match the attributes specified in the authentication-token file for authentication to succeed. Currently the most useful thing to put in this parameter is `pwd`, the challenge password from the request.
`deferOnFailure`	Specifies whether the server should defer CEP requests that fail authentication. • `true` specifies that the server should defer CEP-enrollment requests that fail authentication; the deferred requests get queued for agent approval. • `false` specifies that the server should reject CEP-enrollment requests that fail authentication.

During CEP enrollment, all the attributes in the subject name and the challenge password are passed to the `FlatFileAuth` plug-in. The plug-in looks in a prepared file (referred to as the authentication-token file in this document), which consists of a series of entries for each valid enrollee, to determine if the request should be authenticated. For the Certificate Manager to be able to locate the appropriate entry in the authentication-token file before it does any checking of the password, you must identify attributes that are unique in each router request. You do this by setting the `keyAttributes` parameter of the `FlatFileAuth` plug-in implementation to the list of attributes which will be unique in the CEP request.

Table 27.3 lists the default attributes set by the router in the request.

Table 27.3 Default request attributes set by the router

Attribute	Description
UNSTRUCTUREDNAME	Specifies the DNS name of the router (for example, `router32.siroe.com`). This is always specified in the request.
UNSTRUCTUREDADDRESS	Specifies the IP address of the router (for example, `101.22.33.124`). This may not be in the request—a user may not want to include this in the subject name of the router certificate, and hence choose not to specify one during enrollment.
SERIALNUMBER	Specifies the serial number of the router (for example, `239333`). This can sometimes be found on a label on the back of the router. It is also available by typing the show version command. This may not be in the request—a user may not want to include this in the subject name of the router certificate, and hence choose not to specify one during enrollment.

You can identify one or more of these attributes as unique attributes in the authentication-token file. For most cases, specifying the `UNSTRUCTUREDNAME` as the unique attribute will suffice. In this case, you would set the `keyAttributes` parameter as follows:

```
auths.instance.flatfile.keyAttributes=UNSTRUCTUREDNAME
```

However, if this is not unique, you may specify both `UNSTRUCTUREDNAME` and `UNSTRUCTUREDADDRESS` as unique attributes. In this case, you would set the `keyAttributes` parameter as follows:

```
auths.instance.flatfile.keyAttributes=UNSTRUCTUREDNAME,UNSTRUCT
UREDADDRESS
```

This will force the server to use both these attributes to locate an entry in the authentication-token file. Note that both the attributes must be present in the request for authentication to succeed.

There's an added advantage in determining unique attributes for it allows you to enforce a rule on the attributes that must be present in the CEP enrollment request. For example, if you would like to enforce that a particular router be assigned to an IP address and host name, you could set the `keyAttributes` parameter as follows:

```
auths.instance.flatfile.keyAttributes=UNSTRUCTUREDNAME,UNSTRUCT
UREDADDRESS,SERIALNUMBER
```

Once an entry has been found in the authentication-token file, the server tests the authentication tokens specified in the `authAttributes` parameter against those in the file. Only if they all match, the server grants the request. For the purposes of this discussion, let us assume that you define a single authentication token named `pwd` for the challenge password. In this case, you would set the `authAttributes` parameter as follows:

```
auths.instance.flatfile.authAttributes=pwd
```

In summary, to implement the automated CEP enrollment process, you need to do the following:

1. Decide on authentication credentials for users.

   Prepare a list of your CEP enrollees and assign a password to each enrollee.

2. Prepare the authentication-token file with the credentials.

   Create a text file with CEP-enrollee information. The format of the authentication-token file must be as follows:

   ```
   <attribute>: <value>
   <attribute>: <value>
   ...
   <attribute>: <value>
   <attribute>: <value>
   ```

   Each enrolling user is represented by a sequence of attribute-value pairs, terminated by a blank line or end-of-file (EOF). The attributes can be any part of the subject name from the request, for example `SERIALNUMBER`, CN, `OU`, `UID`, or the challenge password (`pwd`). An example is shown below:

   ```
   DN: <DN_for_user1>
   UNSTRUCTUREDNAME: router32.siroe.com
   UNSTRUCTUREDADDRESS: 101.22.33.124
   SERIALNUMBER: 239333
   ```

```
pwd: ff93Kd

DN: <DN_for_user1>
UNSTRUCTUREDNAME: router33.siroe.com
UNSTRUCTUREDADDRESS: 101.22.33.125
SERIALNUMBER: 233455
pwd: 35pww3a
```

Note that if you specify a DN for a CEP enrollee in the authentication file, the Certificate Manager replaces the subject name requested by that user (router or VPN client) with the one specified in the file.

After you've created entries for all the CEP enrollees, save the file. Also note the complete path to the file; you'll be required to specify this in the next step.

3. Configure the Certificate Manager to use the `FlatFileAuth` plug-in to verify predetermined credentials.

   Once you have created the authentication file, you should register the `FlatFileAuth` plug-in implementation in the CMS authentication framework and create an instance (named `flatfile`) of the registered plug-in. Be sure to specify the full path to your authentication file and save your changes.

4. Restart the Certificate Manager.

   After changing the configuration file, you must restart the server for the changes to take effect. If the server fails to start, check the log file for any error messages.

5. Provide users with the predetermined authentication credentials.

   Send the password you determined to the CEP enrollees, asking them to enter it when they are prompted for the challenge password during CEP enrollment.

Step 4. Set Up Multiple CEP Services

This step is optional.

By default, the CEP service runs on this URL: `/cgi-bin/pkiclient.exe`

It is possible to set up multiple instances of CEP, each with a different configuration, each listening on a different URL. This is useful if you have different requirements for different types of users. For example, you might want to have one CEP service that authenticates routers and publishes their certificates to the directory and another CEP service that authenticates VPN clients but does not publish their certificates to the directory.

To set up multiple CEP services, use the following example as a guide:

```
## Router configuration
   eeGateway.cep.cep1.appendDN=O=*BASE_DN*
   eeGateway.cep.cep1.createEntry=true
   eeGateway.cep.cep1.entryObjectClass=cep
   eeGateway.cep.cep1.url=/cgi-bin/pkiclient.exe
   eeGateway.cep.cep1.authName=flatfile_router

## VPN configuration
   eeGateway.cep.cep2.url=/vpnenroll
   eeGateway.cep.cep2.authName=flatfile_VPN

## Router authentication parameters in the configuration file
   auths.instance.flatfile_router.fileName=<full_path_to_the_au
   thentication_file>
   auths.instance.flatfile_router.authAttributes=pwd
   auths.instance.flatfile_router.keyAttributes=UNSTRUCTUREDNAM
   E
   auths.instance.flatfile_router.pluginName=flatfile
   auths.instance.flatfile_router.deferOnFailure=true

## VPN authentication parameters in the configuration file
   auths.instance.flatfile_VPN.fileName=<full_path_to_the_authe
   ntication_file>
   auths.instance.flatfile_VPN.authAttributes=pwd
   auths.instance.flatfile_VPN.keyAttributes=CN,OU,O
   auths.instance.flatfile_VPN.pluginName=flatfile
   auths.instance.flatfile_VPN.deferOnFailure=false

## FlatFileAuth plugin registered in the configuration file
```

```
auths.impl.flatfile.class=com.netscape.certsrv.authenticatio
n.FlatFileAuth
```

When setting up multiple CEP services, you can use the `cepsubstore` attribute to differentiate one CEP service from another. For example, if you're setting up separate CEP services for router and VPN-client certificates and want to set different extensions in these certificates, you can make that happen with the help of predicates; see Table 16.2 on page 493.

Certificate Issuance to Routers or VPN Clients

In general, issuing a certificate to a router involves the following steps:

- Step 1. Find the Required Information
- Step 2. Generate the Key Pair for the Router
- Step 3. Request the CA's Certificate
- Step 4. Submit the Certificate Request to the CA

Step 1. Find the Required Information

- Decide whether you want to submit the certificate request for your router to the Certificate Manager (CA) directly or through a Registration Manager.
- Open Netscape Console, and locate the CMS instance that corresponds to the subsystem of your interest. Make sure that the Certificate Manager is started. If you are planning to submit the request via a Registration Manager, make sure that both the Registration Manager and Certificate Manager (to which the Registration Manager forwards its requests) are started.
- Open the CMS window, and verify whether the HTTP port is enabled. If it isn't, enable it; see "Configuring Port Numbers" on page 158. If you are requesting the certificate for an earlier version of router software, make sure that the HTTP port number is set to 80; earlier versions of router software can only connect to port 80.
- Note the CEP enrollment URL. It is in this form:

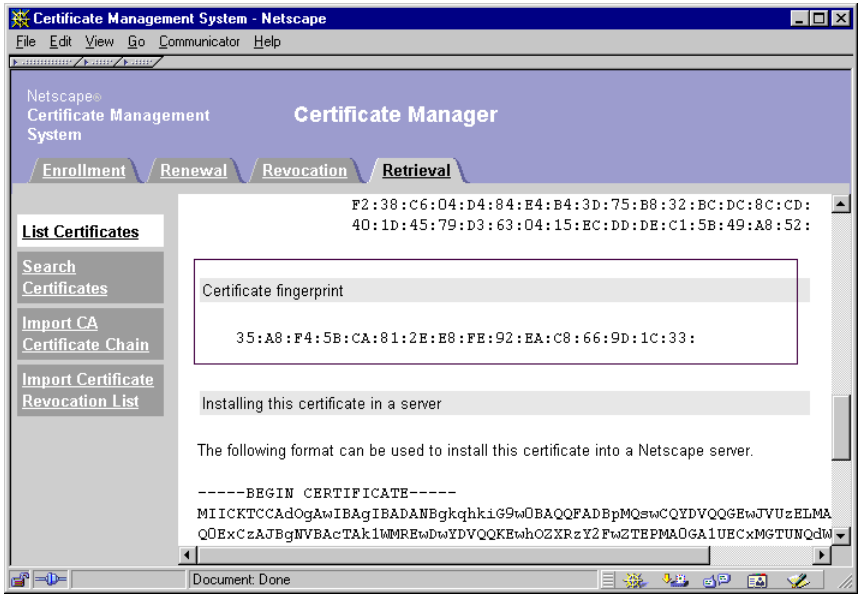
`http://<host_name>:<HTTP_port>/cgi-bin/pkiclient.exe`

where `<host_name>` is in the `<machine_name>.<your_domain>.<domain>` form.

- Note or print the certificate fingerprint information of the Certificate Manager *CA signing certificate*. You will be required to compare this with the fingerprint the router will show on the screen.

  To locate the fingerprint information:

  1. Go to the end-entity page hosted by the Certificate Manager.
  2. Click the Retrieval tab.
  3. List or search for the CA signing certificate.
  4. Click Details.
  5. Scroll down to the section that says "Certificate fingerprint."



- In your router documentation, locate the information specific to requesting certificates for routers. Check the signing algorithm, such as RSA or DSA, and key lengths, such as 512 and 1024, supported by the router. Based on that information, determine the signing algorithm and the key length for the certificate you want to request.
- Find out the password that enables you to access the router in *privileged* mode.
- In your router documentation, locate instructions for requesting certificates. You will be required to run the appropriate commands using this documentation.

Step 2. Generate the Key Pair for the Router

Run the appropriate commands for your router, and generate the key pair. You will be required to provide the signing algorithm, such as RSA or DSA, and the key length, such as 512 or 1024. The longer the key length, the more time the router takes to generate the key pair.

Step 3. Request the CA's Certificate

In this part of the operation, you identify the CA to the router, thus enabling the router to authenticate the CA from which it will request the certificate. You also verify whether the router is talking to the right CA; you do this manually.

Here's what you should do:

1. Run the appropriate command to get the CA certificate.

   The command will ask you to specify the following:

   - An identity for the CA. You can give any identity; choose something you will remember, since you will be required to provide it when you submit the certificate request.
   - The CA's enrollment URL; this is the enrollment URL you identified in Step 1.

2. The router gets the CA certificate and displays its fingerprint on your screen.
3. Verify the fingerprint on your screen with the one you noted down in Step 1.

If it matches, the router is talking to the right CA.

Step 4. Submit the Certificate Request to the CA

To submit the certificate request to the CA:

1. Run the appropriate command.

   The command will ask you for certain information:

   - The CA's identity. You specified this in Step 3.
   - Challenge password. If you enter one, write it down; you will be required to specify this password to revoke the certificate.
   - The CEP enrollment URL.
   - Whether you want to include the router's serial number in the request. If you choose to include the serial number, it will be included in the certificate's subject name.
   - Whether you want to include the router's IP address in the request. If you choose to include the IP address, it will be included in the certificate's subject name.

2. This step depends on your CA's configuration for router enrollment.

   - If the CA to which the router submitted the request employs automatic enrollment (or authentication) for routers, the request will get processed by the CA. The CA may return the certificate to the router in the same transaction. If it doesn't, the router checks with the CA at periodic intervals; in the router configuration you can specify how often the router should poll the CA for the certificate and how many attempts it should make. By default, the router checks the CA every minute.
   - If the CA to which you submitted the request is configured for manual enrollment (or authentication), the request gets queued and awaits approval by an agent.

Important Your router may require additional configuration changes. Be sure to follow the information in your router documentation.

Example

The example below shows the commands and associated outputs for a Cisco router:

```
# To perform certificate enrollment for a router using CEP, you must be
# in privileged mode, which you do by typing “enable” first, and then
# entering the password.
   router> enable
   router% config terminal

   router(config)#crypto key generate rsa
   The name for the keys will be: netscape.mcom.com
   Choose the size of the key modulus in the range of 360 to 2048 for
      your General Purpose Keys. Choosing a key modulus greater than 512
      may take a few minutes.

   How many bits in the modulus [512]:
   Generating RSA keys ...
   [OK]

   router(config)#crypto ca identity test-ca
   router(ca-identity)#enrollment url http://ca-hostname.domain.com/
                                        cgi-bin/pkiclient.exe
   router(ca-identity)#exit

   router(config)#crypto ca authenticate test-ca
   Certificate has the following attributes:
   Fingerprint: 24D34656 EB830C39 DD9E8179 0A4EBA98
   % Do you accept this certificate? [yes/no]: yes

   router(config)#crypto ca enroll test-ca
   %
   % Start certificate enrollment ..
   % Create a challenge password. You will need to verbally provide this
      password to the CA Administrator in order to revoke your
      certificate. For security reasons your password will not be saved
```

```
   in the configuration. Please make a note of it.

Password:
Re-enter password:

% The subject name in the certificate will be: router.domain.com
% Include the router serial number in the subject name? [yes/no]: yes
% The serial number in the certificate will be: 08342063
% Include an IP address in the subject name? [yes/no]: yes
Interface: ethernet0
Request certificate from CA? [yes/no]: yes
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the
   fingerprint.

router(config)# exit

router#show crypto ca certificates
CA Certificate
 Status: Available
 Certificate Serial Number: 1
 Key Usage: Not Set

Certificate
 Subject Name
    Name: netscape.mcom.com
    IP Address: 208.12.63.193
    Serial Number: 08342063
 Status: Pending
 Key Usage: General Purpose
 Fingerprint:  91D70D7F D8BF0DFA E13F00B0 6EA706A0 00000000
```

Certificate Renewal

Every certificate issued by Certificate Management System has a validity period that determines its expiration date. The validity period of a certificate is determined by the *validity constraints* policy settings at the time the certificate was issued (see "Validity Constraints Policy" on page 543). For a certificate to be valid beyond its expiration date, it must be renewed. Otherwise, the certificate becomes invalid, and the entity owning the certificate will no longer be able to use it. Also, the expired certificate will take up space in your publishing directory and in the internal database of Certificate Management System.

Note The Job scheduler component of Certificate Management System enables you to schedule a *job* for removing expired certificates from the publishing directory. For details, see "Directory Update and Notification" on page 444.

Renewal of Client Certificates

Certificate Management System allows end users to renew their certificates by using the certificate renewal form hosted by a Certificate Manager or Registration Manager; the form is available in the Renewal tab of the HTTPS interface. End users can use this form for renewing a single certificate or dual certificates. To renew a certificate, end users need to present their current certificate (for SSL client authentication) during renewal. For renewal purposes, the server accepts even expired certificates.

Certificate Management System comes with a policy plug-in module that allows you to configure both Certificate Manager and Registration Manager to apply specific renewal rules for certificates; you can specify how long a renewed certificates should be valid and how many days before expiration of the current certificate an end user can renew the certificate. The renewal policy rule, if enabled, also enforces that the certificate presented during client authentication is valid or is expired; it cannot have been revoked. For more information about this policy module, see "Renewal Validity Constraints Policy" on page 525.

When a certificate is renewed, Certificate Management System formulates a new certificate with the same public key and other details from the existing certificate; the renewal does not include key changeover. The server, if configured for LDAP publishing, also publishes the new certificate to the publishing directory.

The following steps describe how a Certificate Manager or Registration Manager renews client certificates:

1. The end user uses the certificate renewal form provided in the Renewal tab of the End Entity Services interface of a Certificate Manager or Registration Manager and submits a certificate-renewal request.

   The end user can request renewal of the certificate he or she presents for client authentication or renewal of all certificates (that belong to the end user) with the same subject name as in the authentication certificate. After successful authentication, the server fetches the currently-valid (or expired) certificates for the end user from its certificate repository.

2. The server formulates a renewal request based on the certificate being used for renewal.

3. The server applies its currently configured renewal policies to the request. If the policies require that the request be deferred for agent approval, the server stores the renewal request in the request queue and responds to that end user indicating that the request is waiting for approval by an agent. If the request is rejected by any of the policy rules, the server sends an error response.

4. The server formulates a CRMF request for one or more certificates as the case may be using a suitable template.

As an administrator, you can configure a Certificate Manager or Registration Manager to automatically notify end users to renew their certificates before the current ones expire. If you enable this feature, the subsystem periodically queries the internal database for *already expired* or *about to expire and not renewed already* certificates, and alerts the user and you (optional) to imminent certificate expiration. For more information about this feature, see "Certificate Renewal Notifications" on page 435.

Renewal of Server Certificates

Certificate Management System allows server administrators to renew their certificates by using the server enrollment form hosted by a Certificate Manager or Registration Manager. The renewal process is similar to the enrollment process in that the administrators must manually generate the certificate-signing

request using the server's key pair, paste that request in the manual enrollment form, and submit the request. For details, see "Certificate Issuance to Servers" on page 1081.

Note For renewing the certificates of a Certificate Manager, Registration Manager, or Data Recovery Manager, see "Renewing Certificates for the Subsystems" on page 286.

Certificate Revocation

Certificate Management System allows a certificate to be revoked by an end user (the original owner of the certificate), a server administrator, or by a Certificate Manager or Registration Manager agent. End users can revoke certificates by using the Revocation form provided in the end-entity services interface. Agents can revoke end-entity certificates by using the appropriate form in the Agent Services interface. Certificate-based (SSL client authentication) or challenge-password-based authentication is required in both cases; for details, see "Authentication of End Users During Certificate Revocation" on page 314.

- An end user can revoke only those certificates that contain the same subject name as in the certificate presented for authentication; if using a challenge password, the user can revoke only the certificate that is associated with that password. After successful authentication, the server lists the certificates belonging to the end user. The end user can then select the certificate to be revoked or can revoke all certificates in the list. The end user can also specify additional details, such as the date of revocation and revocation reason for each certificate or for the list as a whole. For instructions on how end users revoke their certificates, see the online help available by clicking the Help buttons on the end-entity forms.

- Agents can revoke certificates based on a range of serial numbers or based on one or more subject name components. Upon submission of the revocation request, the agent receives a list of certificates from which she or he can pick the ones to be revoked. For instructions on how agents revoke end-entity certificates, see *Netscape Certificate Management System Agent's Guide*.

Upon receiving the list of certificates to be revoked, the Registration Manager formulates a CMMF request and sends it to the Certificate Manager. The Certificate Manager marks the corresponding certificate records in its certificate

store (maintained in the internal database) as *revoked* and if configured to do so, removes the revoked certificates from the publishing directory and updates the CRL in the publishing directory.

Chapter

28

Recovering Encrypted Data

When data is stored in encrypted form, you must have the private key that corresponds to the public key that was used to encrypt the data in order to decrypt and read it. If the private key is lost, the data cannot be retrieved. A private key can be lost because of a hardware failure, for example, or because the key's owner forgets the password or loses the hardware token in which the key is stored. Similarly, encrypted data cannot be retrieved if the owner of the key is unavailable to supply it—for example, has left the organization that owns the data.

This chapter explains how to use the Data Recovery Manager to archive users' encryption private keys and how to use the archived keys later, in place of missing encryption keys, to recover encrypted data.

The chapter has the following sections:

PKI Setup for Key Archival and Recovery

To be able to archive users' encryption private keys and recover them later, you need a PKI setup that includes the following elements:

- Clients that can generate dual keys and that support the key archival option (using the CRMF/CMMF protocol)
- An installed and configured Data Recovery Manager
- HTML forms with which your users can request dual certificates (based on dual keys) and key recovery agents can request key recovery

The sections that follow explain these elements in detail. For step-by-step instructions on setting up your PKI environment for key archival and recovery, see "Setting Up Key Archival and Recovery Process" on page 1133.

Clients That Can Generate Dual Key Pairs

Only keys that are used exclusively for encrypting data should be archived; signing keys in particular should never be archived. Having two copies of a signing key would defeat the certainty with which the key identifies its owner; a second copy could be used to impersonate the digital identity of the original key owner.

Clients that generate single key pairs use the same private key for both signing and encrypting data, so you cannot archive and recover a private key deriving from a single key pair. By contrast, clients that can generate dual key pairs use one private key for encrypting data and the other for signing data. Because the encryption private key is separate, you can archive it.

In addition to generating dual key pairs, your users' clients must also support the encryption key archival option in certificate requests. This option triggers the key archival process at the time encryption private keys are generated as a part of certificate issuance.

Netscape Communicator versions 4.7 and later support dual key-pair generation when used in conjunction with Netscape Personal Security Manager (Personal Security Manager). For a brief introduction to Personal Security Manager, see page 51.

Data Recovery Manager

With the Data Recovery Manager, you can archive data encryption keys when they are created during dual key-pair generation. You can then recover the keys if they are lost or the key owner is unavailable.

The Data Recovery Manager can archive and recover keys only from clients that support dual key-pair generation and the key archival option in certificate requests.

Certificate Management System does not provide any policy plug-in modules for the Data Recovery Manager. However, you can write custom policy plug-in modules (that is, write Java classes that implement these rules), register them in the Data Recovery Manager's policy framework, and create policy rules using these plug-in implementations.

Forms for Users and Key Recovery Agents

End users' encryption private keys are archived by the Data Recovery Manager when they are generated. So, for key archival to occur, the enrollment form that users fill out to request dual certificates must have the JavaScript code for activating the key archival process embedded in it, along with a valid copy of the Data Recovery Manager's transport certificate. Then, when a Certificate Manager or Registration Manager that is processing the user's certificate issuance request detects the key archival option, it automatically requests the service of the Data Recovery Manager. For information on customizing this form, see "Step C. Customize the Certificate Enrollment Form" on page 1136.

Initiating the key recovery process also requires its own HTML form. By default, the Data Recovery Manager Agent Services interface provides a form for initiating the process and retrieving keys. For information on customizing this form, see "Step D. Customize the Key Recovery Form" on page 1144.

Key Archival Process

If your certificate infrastructure has been set up for key archival, the Data Recovery Manager automatically archives users' encryption private keys. For general information on the type of PKI setup needed for archiving keys, see "PKI Setup for Key Archival and Recovery" on page 1116. For specific instructions on setting up a key archival and recovery infrastructure, see "Setting Up Key Archival and Recovery Process" on page 1133.

Why You Should Archive Keys

If a user loses a private data-encryption key or is unavailable to use his or her private key, the key must be recovered before any data that was encrypted with the corresponding public key can be read. You can recover the private key if an archival copy of it was created when the key was generated.

Here are a few situations in which you might need to recover a user's encryption private key:

- An employee loses the encryption private key (for example, after a disk crash or by forgetting the password to the key file) and cannot read encrypted mail messages.
- An employee is on an extended leave, and you need access to an encrypted document in his or her files.
- An employee leaves the company, and company officials need to perform an audit that requires gaining access to the employee's encrypted mail.

Where the Keys are Stored

If configured properly, the Data Recovery Manager, stores your users' encryption private keys automatically whenever the associated or connected Registration Manager or Certificate Manager issues certificates to your users. The Data Recovery Manager stores encryption private keys in a secure key repository in its internal database; each key is stored as a key record.

The archived copy of the key remains encrypted (or wrapped) with the Data Recovery Manager's storage key; see "Storage Key Pair" on page 233. It can be decrypted (or unwrapped) only by using the corresponding private key, to which no individual has direct access. A combination of one or more key recovery agents' passwords enables the Data Recovery Manager to retrieve its private storage key and use it to decrypt and recover an archived key. For details on how this process works, see "Key Recovery Agents and Their Passwords" on page 1122.

The Data Recovery Manager indexes stored keys by key number (or ID), owner name, and a hash of the public key, allowing for highly efficient searching by name or by public key. The key recovery agents have the privilege to insert, delete, and search for key records. The search feature works like this:

- When the key recovery agents search by the key ID, only the key that corresponds to that ID is returned.
- When the agents search by user name, all stored keys belonging to that owner are returned.
- When the agents search by the public key in a certificate, only the corresponding private key is returned.

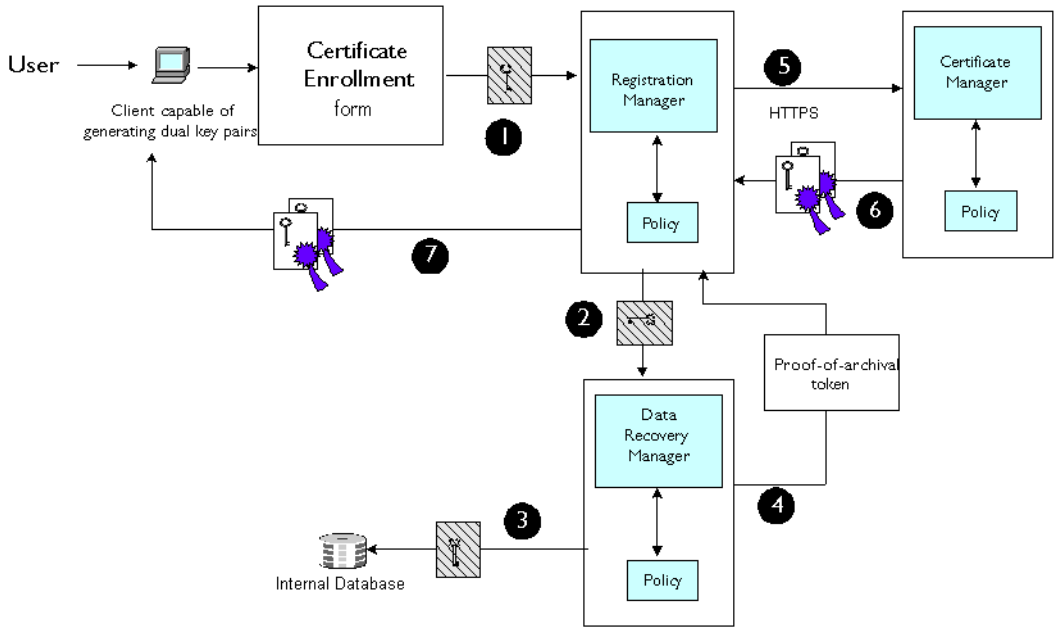
How Key Archival Works

When a Certificate Manager or Registration Manager receives a certificate request that contains the key archival option, it automatically requests the service of the Data Recovery Manager to archive the user's encryption private key. The Data Recovery Manager receives an encrypted copy of the user's private key and stores the key in its key repository. To archive the key, the Data Recovery Manager uses two special key pairs:

- A transport key pair and corresponding certificate
- A storage key pair

Figure 28.1 illustrates how the key archival process occurs when a user requests a certificate. The deployment scenario shown in this figure has a Registration Manager acting as the trusted enrollment authority to a Certificate Manager and Data Recovery Manager.

Figure 28.1 How the key archival process works



These are the steps shown in Figure 28.1:

1. A user uses a client capable of generating dual key pairs to access the certificate enrollment form served by the Registration Manager, fills in all the information, and submits the request.

   The Registration Manager detects the key archival option in the user's request and asks the client for the user's encryption private key.

   The client encrypts the user's encryption private key with the public key from the Data Recovery Manager's transport certificate; a copy of the transport certificate is embedded in the enrollment form.

2. Upon receiving the encrypted key from the client, the Registration Manager sends it to the Data Recovery Manager for storage, along with some other information (including the user's public key). Then, the Registration Manager waits for verification from the Data Recovery Manager that the private key has been received and stored and that it corresponds to the user's public encryption key.

3. Upon receiving the encrypted key from the Registration Manager, the Data Recovery Manager decrypts it with the private key that corresponds to the public key in its transport certificate. After confirming that the private encryption key corresponds to the user's public encryption key, the Data Recovery Manager encrypts it again with its storage key before storing it in its internal database. (The storage key either resides in a software or a hardware token and is never exposed to any other entity.)

4. Once the user's private encryption key has been successfully stored, the Data Recovery Manager uses the private key of its transport key pair to sign a token confirming that the key has been successfully stored; the Data Recovery Manager then sends the token to the Registration Manager.

5. After the Registration Manager receives and verifies the signed token, it sends the certificate request to the Certificate Manager for issuance.

6. The Certificate Manager formulates two certificates, one each for signing and encryption key pairs, and returns them to the Registration Manager.

7. The Registration Manager forwards the certificates to the client (the user).

Note that all three subsystems subject the request to configured policy rules at appropriate stages. If the request fails to meet any of the policy rules, the subsystem rejects the request.

Key Recovery Process

The Data Recovery Manager supports agent-initiated key recovery. In this method of key recovery, designated recovery agents use the Key Recovery form provided in the Data Recovery Manager Agent Services interface to process key recovery requests, list archived keys, and approve recovery. With the approval of a specified number of agents, an organization can recover keys when the key's owner is unavailable or when keys have been lost.

Key Recovery Agents and Their Passwords

Key recovery agents have the authority to retrieve end users' encryption private keys. The recovery agent's role can be performed by any person in your organization. As system administrator, you can designate one or more individuals to be key recovery agents. These individuals need to do the following:

- They must specify a secure password, which in combination with other recovery agents' passwords will be used for protecting the database in which the Data Recovery Manager stores users' keys. You facilitate this by allowing each recovery agent to enter a password in the Data Recovery Manager configuration.

- They must be available to retrieve your users' encryption private keys if the need arises. It isn't necessary for all key recovery agents to be available for the key recovery operation. You specify how many agents are required to authorize the recovery of a key; see "Key Recovery Agent Scheme" on page 1129. However, the specified number of key recovery agents must all provide their passwords at the same time to authorize the recovery of a specific key.

The first time you create key recovery agents and specify their passwords is during the installation of the Data Recovery Manager. However, you can change the number of recovery agents and their passwords later by modifying it in the Data Recovery Manager configuration; see "Changing Key Recovery Agents' Passwords" on page 1132.

Secret Sharing of Storage Key Password

The Data Recovery Manager uses the private key of its *storage key pair* to encrypt the repository where it store users' encryption private keys. This requires that the storage key be well protected. For the protection of the storage key pair, the Data Recovery Manager supports a password-splitting mechanism called *m of n secret splitting or sharing*, whereby it splits the PIN that protects the token in which the storage key pair resides among *n* number of key recovery agents and reconstructs the PIN only if *m* number of recovery agents provide their individual passwords; *n* must be an integer greater than 1 and *m* must be an integer less than or equal to *n*.

Here's how the *m* of *n* secret splitting mechanism gets built and works:

During the installation of a Data Recovery Manager, you generate the storage key pair and specify the hardware token in which the key pair is to be stored. At this time, you also specify a PIN (or password) to protect the token, the total number of key recovery agents (*n*), and how many of these agents (*m*) are required to perform a key recovery operation. You can change the *m* of *n* secret splitting later; for details, see "Key Recovery Agent Scheme" on page 1129.

The Data Recovery Manager splits the PIN for the token into *n* parts or pieces. It then encrypts these parts with the passwords that are provided by the authorized key recovery agents.

During the key recovery procedure, the required number of key recovery agents (*m*) provide their identifiers and passwords. After verifying the passwords, the Data Recovery Manager reconstructs the PIN for the token based on the given information.

Interface for the Key Recovery Process

With the Key Recovery form provided in the Data Recovery Manager Agent Services interface, key recovery agents can collectively unlock the key repository of the Data Recovery Manager and retrieve end users' encryption private keys and associated certificates in a PKCS #12 package, which can then be imported into the client. For an overview of this process, see "How Agent-Initiated Key Recovery Works" on page 1126.

Because key recovery agents use the Data Recovery Manager Agent Services interface, agent-initiated key recovery invariably involves the Data Recovery Manager agent and key recovery agents. The Data Recovery Manager agent's certificate is required to access the Key Recovery form, and key recovery agents' passwords are required to unlock the key repository. For information on Data Recovery Manager agents, see "Agents" on page 173.

Your organization's PKI policy may require that the key recovery process be restricted to authorized recovery agents only, preventing any Data Recovery Manager agent from being involved. If so, you should ask all key recovery agents to get client certificates and set them up as Data Recovery Manager agents. For instructions, see "Setting Up Agents" on page 193.

Local Versus Remote Key Recovery Authorization

Key recovery agents can authorize the recovery of a key locally or remotely. The overview of local and remote authorization provided in this section is intended to help you determine which to use for your organization. You may find it useful to take a look at the Data Recovery Manager agent-specific information in the *Netscape Certificate Management System Agent's Guide*. To locate online versions (HTML and PDF) of this guide, check this file:

```
<server_root>/manual/index.html
```

Local Key Recovery Authorization

To initiate key recovery locally, the required number of recovery agents assemble in front of the host system that allows them to access the Data Recovery Manager Agent Services interface. Either a Data Recovery Manager agent or a key recovery agent with a Data Recovery Manager agent certificate accesses the Key Recovery form hosted by the Data Recovery Manager and initiates the key recovery process. All key recovery agents enter their IDs and passwords on the same Recovery Authorization form presented by the Data Recovery Manager. If the information presented is correct, the Data Recovery Manager retrieves the requested key and returns it along with the corresponding certificate in the form of a PKCS #12 package.

By default, key recovery authorization is local.

Remote Key Recovery Authorization

To authorize key recovery remotely, the required number of recovery agents access the Data Recovery Manager Agent Services interface at their own locations and use the Authorize Recovery button to enter each authorization separately.

Before key recovery agents can authorize key recovery remotely, they must be set up to function as Data Recovery Manager agents. This role gives them the privilege to access the Data Recovery Manager's Agent Services interface directly.

In remote key recovery authorization, one of the key recovery agents informs all required recovery agents about an impending remote key recovery process. All recovery agents access the Key Recovery page hosted by the Data Recovery Manager. One of the agents initiates the key recovery process. The Data Recovery Manager returns a notification to each agent. The notification includes

a recovery authorization reference number identifying the particular key recovery request that the agent is required to authorize. Each agent uses the reference number and authorizes key recovery separately.

The Data Recovery Manager informs the agent who initiated the key recovery process of the status of the authorizations. When all of the authorizations are entered, the Data Recovery Manager checks the information. If the information presented is correct, it retrieves the requested key and returns it along with the corresponding certificate in the form of a PKCS #12 package to the agent who initiated the key recovery process.

Key recovery agents can switch to remote authorization by deselecting the local authorization option in the Key Recovery form.

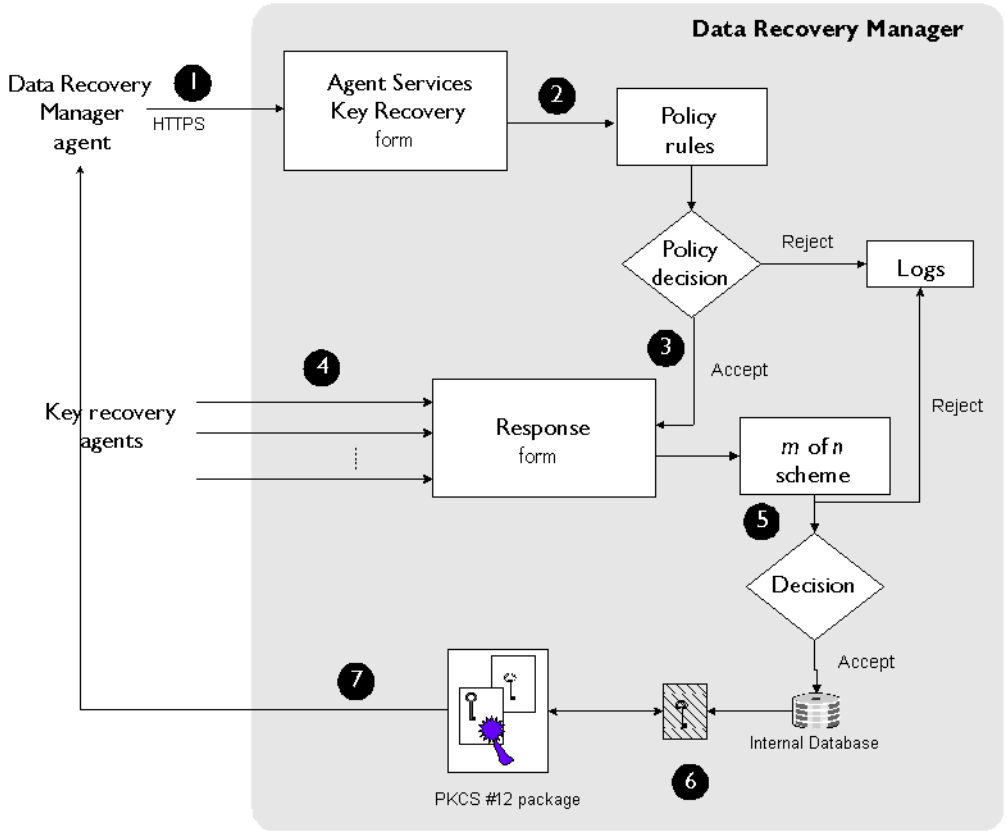
How Agent-Initiated Key Recovery Works

In an agent-initiated key recovery, the key is recovered by the collective efforts of a Data Recovery Manager agent and authorized key recovery agents. You may need to resort to this type of key recovery if the owner of a key cannot be reached and the authorities in the organization need to access that user's encrypted data (for example, S/MIME mail messages).

Upon retrieving the private encryption key (in the form of a PKCS #12 package), the agents may forward the key to the original user, the manager of the original owner, or some other authorities.

Figure 28.2 illustrates how agent-initiated key recovery works.

Figure 28.2 The agent-initiated key recovery process



These are the steps shown in Figure 28.2:

1. The Data Recovery Manager agent accesses the Key Recovery form using the appropriate client certificate, types the identification information pertaining to the person whose encryption private key needs to be recovered, and submits the request.

   The request is submitted to the Data Recovery Manager over HTTPS.

2. The Data Recovery Manager subjects the key recovery request to its policy checks.

3. If the request passes all the policy rules, the Data Recovery Manager sends a confirmation HTML page to the web browser the agent used. If the request fails any of the policy checks, the server logs an appropriate error message.

   The confirmation page contains information and input sections:

   - The information section includes the user's information.
   - The input section includes fields for entering the user's certificate corresponding to the key that needs to be recovered, the password for the PKCS #12 package, and key recovery agents' passwords.

     The Data Recovery Manager uses the certificate to construct the PKCS #12 package (which includes the user's encryption private key and corresponding certificate), the PKCS #12 password to encrypt the PKCS #12 package, and key recovery agents' passwords to construct the PIN required to unlock its key repository.

4. The key recovery agents verify the information in the confirmation page and enter the certificate in MIME-64 format, the password for the PKCS #12 package, and their individual identifiers and passwords. The Data Recovery Manager agent submits the page to the Data Recovery Manager.

5. The Data Recovery Manager matches the key recovery agent information with its *m of n scheme* (see "Key Recovery Agent Scheme" on page 1129). After verifying that the required number of recovery agents entered their passwords, the server uses the agents' passwords to construct the PIN required to access the private key repository.

6. The Data Recovery Manager then retrieves the user's private key from its key repository and decrypts it by using the private component of the storage key pair.

7. The Data Recovery Manager packages the user's certificate and the corresponding private key as a PKCS #12 package and encrypts it with the PKCS #12 password provided by the recovery agent. It then delivers the package to the client the recovery agent used to initiate the key recovery process, and prompts the agent to store the encrypted package. The agent may choose to store the package in the local file system of the client machine (only if it has restricted access) or on a floppy diskette.

The recovery agent can then send the encrypted PKCS #12 package and the corresponding password to an individual by any secure, out-of-band means.

Important The PKCS #12 package contains the private key. To minimize the risk of key compromise, the recovery agent must use any secure, out-of-band means to deliver the PKCS #12 package and password to the key recipient. As an administrator, you should recommend the recovery agent to use a good password for encrypting the PKCS #12 package, and also consider setting up an appropriate delivery mechanism.

Key Recovery Agent Scheme

The *key recovery agent scheme* consists of configuring the Data Recovery Manager to recognize a fixed number of key recovery agents (a minimum of one) and specifying how many of these agents are required to authorize a key recovery request before the archived key is restored. Each recovery agent provides the Data Recovery Manager with a password, which it uses to generate a unique PIN; the Data Recovery Manager uses the PIN to protect its storage key pair, which in turn protects users' keys.

The Data Recovery Manager tracks the key recovery agent password for each agent and allows you to facilitate changing agents' passwords; you do not have direct access to these passwords or the actual storage key password. Each password retrieves only a part of the private storage key.

You first specified the key recovery agent scheme when you installed the Data Recovery Manager.

Changing the Key Recovery Agent Scheme

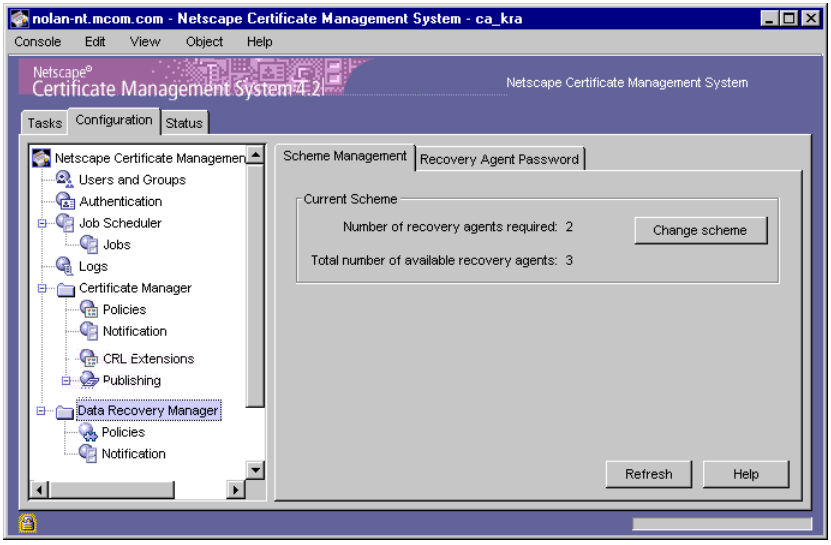
You can change the total number of key recovery agents for a Data Recovery Manager and the number of key recovery agents required to retrieve an end user's encryption private key from the Data Recovery Manager's key repository.

To change the key recovery agent scheme:

1. Access the CMS window (see "Logging In to the CMS Window" on page 78).
2. Click the Configuration tab.

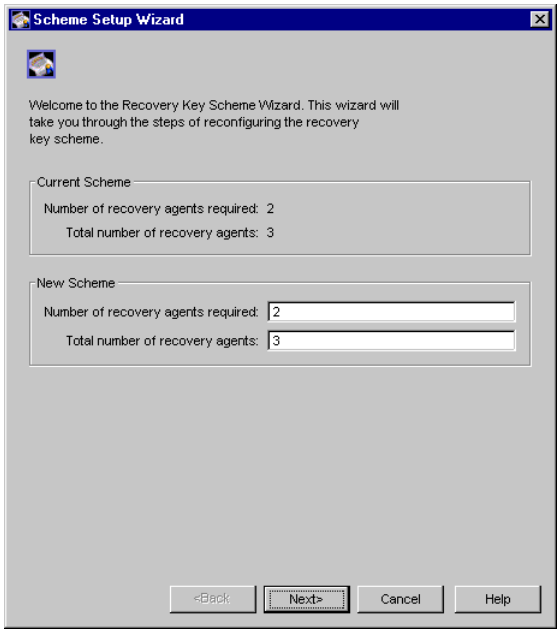
3. In the navigation tree, select the Data Recovery Manager, and in the right pane, click the Scheme Management tab.

   The Scheme Management tab shows the current key recovery scheme.



4. Click Change scheme.

The Change Recovery Key Scheme window appears.



5. In the New Scheme section, make the appropriate changes:

   Number of recovery agents required. Type the number of agents required to authorize a key recovery process. The number cannot be zero and must be equal to or less than the total number of recovery agents.

   Total number of recovery agents. Specify the total number of key recovery agents. The number cannot be less than one and must be equal to or greater than the number of agents required to authorize the key recovery operation.

6. Click Next.

7. For each agent, enter a user name and password, then click Next.

   The number of screens depends on the total number of agents you have specified.

8. When you have entered all agent information, click Finish.

   You are returned to the Scheme Management tab.

Changing Key Recovery Agents' Passwords

As administrator, you have the responsibility of safeguarding the security of each Data Recovery Manager installed in your PKI setup. One of the safety measures you can implement is to ask your key recovery agents to change their passwords periodically. This way, you will be sure that the required number of agents are available if a key needs to be recovered. If key recovery agents routinely change their passwords, they are less likely to forget them.

The CMS window allows you to view the list of currently designated key recover agents and, if necessary, change an agent's password.

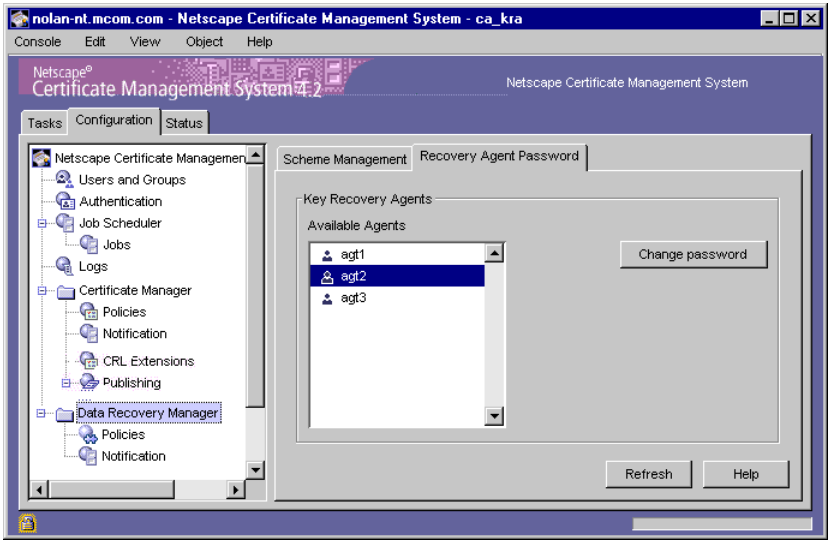
To change key recovery agents' passwords:

1. Access the CMS window (see "Logging In to the CMS Window" on page 78).

2. Click the Configuration tab.

3. In the navigation tree, select the Data Recovery Manager, and in the right pane, click the Recovery Agent Password tab.

   The tab shows current key recovery agents in the Available Agents list.



4. Select the agent whose password needs to be changed, and click Change Password.

The Change Password dialog box appears.

5. Allow the agent to enter the appropriate information.

   During installation, the Data Recovery Manager prompts you to enter key recovery agent passwords (by default, they are set to `agent<n>`, where `<n>` can be 1, 2, and so on, depending on the number of agents). The number of passwords you must enter depends on the key recovery agent scheme you chose; for details, see "Key Recovery Agent Scheme" on page 1129. If you are changing a password for the first time after installation, in the Old password field you must enter the recovery agent password you specified during installation. Then in the remaining fields, allow the key recovery agent to enter the new password information. If you have more than one key recovery agent, repeat this procedure for all the agents.

   Old Password. Type the current password for the key repository.

   New Password. Type the new password for the key repository.

   Confirm Password. Retype the new password exactly as you typed it in the previous field.

6. Click OK.

   You are returned to the Recovery Agent Password tab.

Setting Up Key Archival and Recovery Process

By default, the Data Recovery Manager is not configured to archive or recover end users' encryption private keys. This section explains how to set up key archival and recovery processes.

- Step 1. Set Up the Key Archival Process

- Step 2. Set Up the Key Recovery Process
- Step 3. Test Your Key Archival and Recovery Setup

Step 1. Set Up the Key Archival Process

Before proceeding with this section, you should have read "Key Archival Process" on page 1118. In particular, you should be familiar with how the key archival process works. If you are not, see "How Key Archival Works" on page 1119.

To set up the key archival process, follow these steps:

- Step A. Deploy Clients That Can Generate Dual Key Pairs
- Step B. Connect the Enrollment Authority and the Data Recovery Manager
- Step C. Customize the Certificate Enrollment Form
- Step D. Configure Key Archival Policies

Step A. Deploy Clients That Can Generate Dual Key Pairs

You can use the Data Recovery Manager to archive and recover keys only from clients that support dual key-pair generation, the key archival option, and the CMC protocol. Clients that do not meet this criteria cannot be used with the Data Recovery Manager. To understand why you need to use clients that can generate dual key pairs, see "Clients That Can Generate Dual Key Pairs" on page 1116. The same section also points you to an introduction to Netscape Personal Security Manager, which when plugged into Netscape Communicator version 4.7 or later enables it to support the CMC protocol and generate dual key pairs.

You may have already installed Personal Security Manager—for example, you might have installed it as an OCSP-compliant client when setting up a Certificate Manager to publish CRLs to an OCSP responder; see "Step 2. Install an OCSP-Compliant Client" on page 859. If you want to install Personal Security Manager, check its release notes for installation instructions:

```
<server_root>/psm101/release_notes.html
```

Step B. Connect the Enrollment Authority and the Data Recovery Manager

Key archival occurs when dual key pairs are generated by the client. The client generates the key pairs when a user requests a certificate by filling out the appropriate certificate enrollment form served by an enrollment authority, which can be either a Certificate Manager or a Registration Manager. When the enrollment authority detects the key archival option in the request, it initiates the key archival process and requests the service of the Data Recovery Manager for archiving the key.

For the enrollment authority to be able to request the service of the Data Recovery Manager, the two subsystems must be configured to recognize, trust, and communicate with each other. When you installed the Data Recovery Manager, you were asked to connect it to a Certificate Manager or Registration Manager. You might have specified some of the configuration information required for the two subsystems to communicate with each other. Also, if the enrollment authority and the Data Recovery Manager are installed in the same CMS instance, certain configurations are done automatically.

However, to ensure that key archival takes place successfully, you must make sure that the Data Recovery Manager is connected to the appropriate enrollment authority. Also verify whether the enrollment authority has been set up as a privileged user, with an appropriate SSL client authentication certificate, in the internal database of the Data Recovery Manager. By default, the Certificate Manager uses its *SSL server certificate* for SSL client authentication, whereas the Registration Manager uses its *signing certificate* for this purpose; for more information, see "Keys and Certificates for the Main Subsystems" on page 226.

Otherwise, follow the instructions in "Setting Up Trusted Managers" on page 201 and set up the enrollment authority as a trusted front end to the Data Recovery Manager.

Step C. Customize the Certificate Enrollment Form

For the enrollment authority to automatically initiate the key archival process at the time key pairs are generated, a certificate request must include the following information:

- The key archival option—this must be included in the certificate enrollment form that your users use to request certificates.

- The Data Recovery Manager's transport certificate—this must also be included in the certificate enrollment form. The Data Recovery Manager uses it to encrypt the user's encryption private key with the public key in the transport certificate before sending the user's key to its key repository. For information about the key repository, see "Where the Keys are Stored" on page 1119.

  Make sure that the transport certificate, in its base-64 encoded format, is embedded in the form. Otherwise, the Data Recovery Manager will fail to archive users' keys.

All the end user enrollment forms provided by Certificate Management System—for example, the directory-based enrollment form (`DirUserEnroll.html`), directory- and PIN-based enrollment form (`DirPinUserEnroll.html`), and manual enrollment form (`ManUserEnroll.html`)—contain the necessary JavaScript code for initiating the key archival process; for details about these forms, see "Forms for Certificate Enrollment" on page 915. If you are using any of these forms for end-user enrollment, make sure to update the `generateCRMFRequest()` JavaScript method. If you plan to use custom enrollment forms for users, be sure to include the required JavaScript code in those forms.

Figure 28.3 shows the default directory-based enrollment form with the information related to the `generateCRMFRequest()` JavaScript method highlighted. Note that the JavaScript method includes parameters for specifying various things. You are required to update the following information only:

- The Data Recovery Manager's transport certificate.
- The algorithm, length, type, and usage for end users' key pairs. When you update this information, the key archival option is automatically set. For information on specifying key type, length, and algorithm, see `generateCRMFRequest()` in *Javascript API for Client Certificate Management*. The document is at this location:

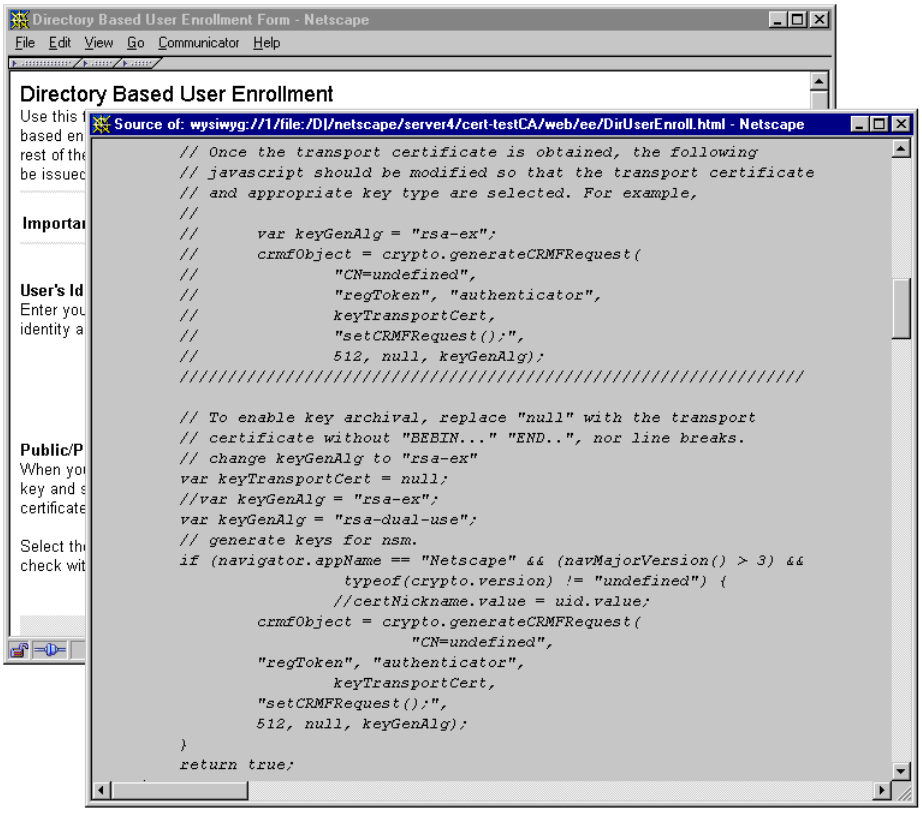
  `<server_root>/psm101/CMCJavascriptapi.html`

  You can also get the latest version of this document form this site:

  `http://docs.iplanet.com/docs/manuals/psm.html`

The steps that follow explain how to do this.

Figure 28.3 Data Recovery Manager's transport certificate information in the enrollment form



1. Copy the transport certificate in its base-64 encoded format

The transport certificate is stored in the Data Recovery Manager's certificate database. If the transport certificate is signed by a Certificate Manager, then a copy of the certificate is also available with the Certificate Manager. Follow the instructions as appropriate.

- To copy the transport certificate information from a Certificate Manager's database:

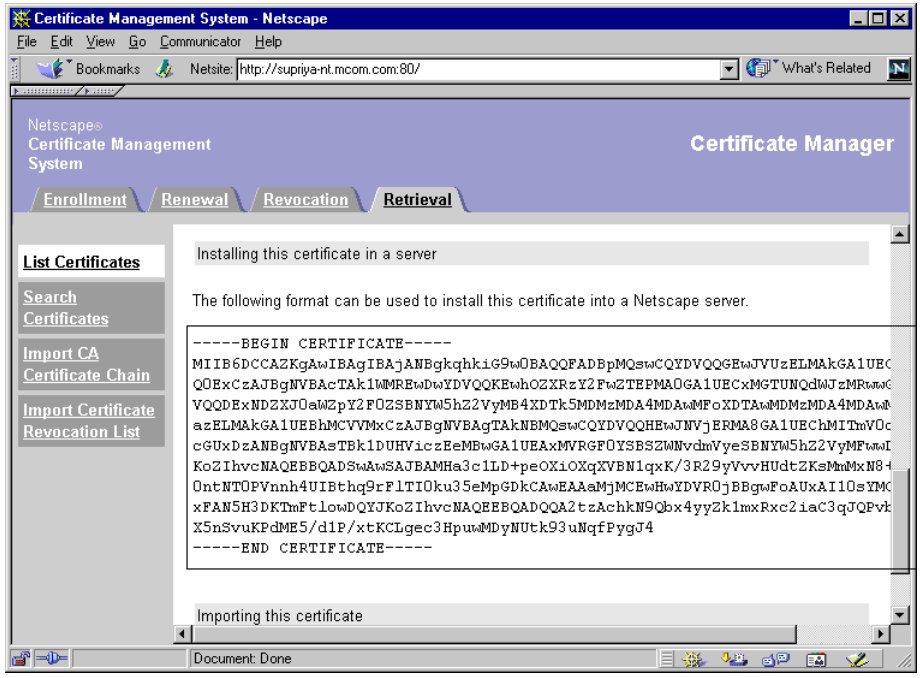
    1. Open a web browser window.
    2. Go to the end-entity page hosted by the Certificate Manager.
    3. Click the Retrieval tab.

4. List or search for the transport certificate.

5. Click Details, and view the certificate information.

   Make sure that the certificate you are looking at is the correct one; the certificate shows the DN that was specified for the transport certificate during the installation of Data Recovery Manager.

6. Scroll down to the section that says "Installing this certificate in a server."



7. Copy the base-64 encoded certificate, *excluding* the marker lines `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`, to a text file. An example is shown below:

```
MIICDjCCAXegAwIBAgICAfMwDQYJKoZIhvcNAQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBA
oTI05ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnBvcmF0aW9uMREwDwYDVQQLEwhIYXJkY29y
ZTEnMCUGA1UEAxMeSGFyZGNvcmUgQ2VydGlmaWNhdGUgU2VydmVyIElJMB4XDTk4MTExOTIzND
IxOVoXDTk5MDUxODIzNDIxOVowLjELMAkGA1UEBhMCVVMxETAPBgNVBAoTCG5ldHNjYXBlMQww
CgYDVQQDEwNLUmEwXDANBgkqhkiG9w0BAQEFAANLADBIAkEArrbDiYUI5SCdlCKKa0bEBn1m83
kX6bdhytRYNkdHB95Bp85SRadmdJV+0OyMxjYAtGCFrmcqEZ4sh2YSov6wIDAQABozYwNDARBg
lghkgBhvhCAQEEBAMCAEAwHwYDVR0jBBgwFoAUl7FtsrYCFlQMl9fjMm3LnNu3oAwDQYJKoZIh
vcNAQEEBQADgYEApvzcUsVIOstaoYSiWb4+aMVH6s1jiJlr5iVHnOKzfsYxPVdUw6uz04AT8N+
1KIarMTKxHPzGAFSLicKLEv4HG4vh6llc86uzRzWpUqqVHgeKN5A8Jyg56D4DkNrXEJ7QdKesA
p13dk5H5qvHelkSPLYYdMXNwNWP
```

- To copy the transport certificate information from a Data Recovery Manager's certificate database:

  1. Open a terminal window in the system that hosts the Data Recovery Manager.

  2. Use the command-line tool called `certutil` to retrieve the transport certificate from the Data Recovery Manager's certificate database. (For information on the `certutil` tool, see Appendix D, "Certificate Database Tool" in the online version of this document.)

     First, go to this directory:
     `<server_root>/cert-<instance_id>/config`

     Next, run this command: `<server_root>/bin/cert/tools/certutil -L -d . -n kraTransportCert cert-<instance_id> -a`

     The transport certificate appears. View the certificate information. Make sure that the certificate you are looking at is the correct one; the certificate shows the DN that was specified for the transport certificate during the installation of Data Recovery Manager.

  3. Copy the base-64 encoded certificate, *excluding* the marker lines `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----`, to a text file. The copied information should look like the example below:

```
MIICDjCCAXegAwIBAgICAfMwDQYJKoZIhvcNAQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVBA
oTI05ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnBvcmF0aW9uMREwDwYDVQQLEwhIYXJkY29y
ZTEnMCUGA1UEAxMeSGFyZGNvcmUgQ2VydGlmaWNhdGUgU2VydmVyIElJMB4XDTk4MTExOTIzND
IxOVoXDTk5MDUxODIzNDIxOVowLjELMAkGA1UEBhMCVVMxETAPBgNVBAoTCG5ldHNjYXBlMQww
CgYDVQQDEwNLUmEwXDANBgkqhkiG9w0BAQEFAANLADBIAkEArrbDiYUI5SCdlCKKa0bEBn1m83
kX6bdhytRYNkdHB95Bp85SRadmdJV+0OyMxjYAtGCFrmcqEZ4sh2YSov6wIDAQABozYwNDARBg
lghkgBhvhCAQEEBAMCAEAwHwYDVR0jBBgwFoAUl7FtsrYCFlQMl9fjMm3LnNu3oAwDQYJKoZIh
vcNAQEEBQADgYEApvzcUsVIOstaoYSiWb4+aMVH6s1jiJlr5iVHnOKzfsYxPVdUw6uz04AT8N+
1KIarMTKxHPzGAFSLicKLEv4HG4vh6llc86uzRzWpUqqVHgeKN5A8Jyg56D4DkNrXEJ7QdKesA
p13dk5H5qvHelkSPLYYdMXNwNWP
```

2. Update the JavaScript method in the enrollment form

To do this:

1. Go to the host system of the enrollment authority and locate the user-enrollment form. The default forms are at this location:

   `<server_root>/cert-<instance_id>/web/ee`

2. Open the enrollment form that you want to use in a text editor.

3. In the form, locate the `generateCRMFRequest()` JavaScript method (see Figure 28.3 on page 1138).

4. Add a variable for the transport certificate.

   Below the commented text, add this line:

   ```
   var kraTransportCert =
   ```

5. Open the text file that has the Data Recovery Manager's transport certificate (the one you copied earlier) and copy the certificate.

6. Paste the certificate as the value of the `kraTransportCert` variable.

   Paste the certificate in front of the `=` sign, remove any line breaks, enclose the certificate within double-quotation marks, and end the string with a semicolon (`;`). When deleting line breaks, be sure not to delete any of the characters in the encoded blob.

   An example is shown below:

   ```
   var kraTransportCert =
   "MIICDjCCAXegAwIBAgICAfMwDQYJKoZIhvcNAQEEBQAwdzELMAkGA1UEBhMCVVMxLDAqBgNVB
   AoTI05ldHNjYXBlIENvbW11bmljYXRpb25zIENvcnBvcmF0aW9uMREwDwYDVQQLEwhIYXJkY29
   yZTEnMCUGA1UEAxMeSGFyZGNvcmUgQ2VydGlmaWNhdGUgU2VydmVyIElJMB4XDTk4MTExOTIzN
   DIxOVoXDTk5MDUxODIzNDIxOVowLjELMAkGA1UEBhMCVVMxETAPBgNVBAoTCG5ldHNjYXBlMQw
   wCgYDVQQDEwNLUmEwXDANBgkqhkiG9w0BAQEFAANLADBIAkEArrbDiYUI5SCdlCKKa0bEBn1m8
   3kX6bdhytRYNkdHB95Bp85SRadmdJV+0OyMxjYAtGCFrmcqEZ4sh2YSov6wIDAQABozYwNDARB
   glghkgBhvhCAQEEBAMCAEAwHwYDVR0jBBgwFoAUl7FtsrYCFlQMl9fjMm3LnNu3oAwDQYJKoZI
   hvcNAQEEBQADgYEApvzcUsVIOstaoYSiWb4+aMVH6s1jiJlr5iVHnOKzfsYxPVdUw6uz04AT8N
   +1KIarMTKxHPzGAFSLicKLEv4HG4vh6llc86uzRzWpUqqVHgeKN5A8Jyg56D4DkNrXEJ7QdKes
   Ap13dk5H5qvHelkSPLYYdMXNwNWP";
   ```

7. Pass the `kraTransportCert` variable to the JavaScript method.

   Replace `null` (the fourth line in the method) with `kraTransportCert`.

8. Specify the key algorithm and key type.

   For information on specifying key type, length, and algorithm, see "generateCRMFRequest()" in *Javascript API for Client Certificate Management*.

   Below is an example that shows how the updated `generateCRMFRequest()` method would look:

   ```
   // generate keys for nsm.
   if (navigator.appName == "Netscape" && (navMajorVersion() >
   ```

```
    3) &&
        typeof(crypto.version) != "undefined") {
        certNickname.value = subject.value;
     crmfObject = crypto.generateCRMFRequest(subject.value,
                  "regToken",
                  "authenticator",
                  kraTransportCert,
                  "setCRMFRequest();",
                  512, null, "rsa-ex",
                  1024, null, "rsa-sign");
}
```

The method triggers the client to generate two RSA key pairs—one key of length 512 for encrypting data and another key of length 1024 for signing data.

9. Save your changes.

Step D. Configure Key Archival Policies

This step is optional.

Unlike Certificate Manager and Registration Manager, no policy plug-in modules are provided for the Data Recovery Manager. If you have implemented any custom policy modules for the Data Recovery Manager's key archival process, you should make sure that they are configured properly. For details on configuring policies for a subsystem, see "Setting up Policy Rules for a Subsystem" on page 689.

Step 2. Set Up the Key Recovery Process

Before proceeding with this section, you should have read "Key Recovery Process" on page 1122. In particular, you should be familiar with how the key archival process works. If you are not, see "How Agent-Initiated Key Recovery Works" on page 1126.

The Data Recovery Manager supports agent-initiated key recovery process, in which end users' encryption private keys are recovered by designated key recovery agents. This section explains how to set up the key recovery process.

To set up agent-initiated key recovery process, follow these steps:

- Step A. Verify the m of n Scheme
- Step B. Facilitate the Key Recovery Agents to Change the Passwords
- Step C. Determine the Authorization Mode for Key Recovery
- Step D. Customize the Key Recovery Form
- Step E. Configure Key Recovery Policies

Step A. Verify the m of n Scheme

During the installation of the Data Recovery Manager, you were asked to specify the total number of key recovery agents (a minimum of one) and the number of agents (of this total) required to authorize a key recovery operation. This combination is called *m* of *n* scheme. For more information about this, see "Key Recovery Agent Scheme" on page 1129.

Verify that the current *m* of *n* scheme is appropriate for your PKI setup. If it isn't, change the scheme following the instructions in "Changing the Key Recovery Agent Scheme" on page 1129.

Step B. Facilitate the Key Recovery Agents to Change the Passwords

During the installation of Data Recovery Manager, after you specified the *m* of *n* scheme, you were also prompted to provide unique passwords for each recovery agent. It is quite likely that you specified these passwords yourself instead of it being done by those individuals who have been designated with

the key recovery agents' role in your organization. Therefore, you must get the designated recovery agents to change the passwords entered during installation.

- To understand the significance of key recovery agents' passwords, see "Key Recovery Agents and Their Passwords" on page 1122.
- To get the recovery agents to change the passwords, follow the instructions in "Changing Key Recovery Agents' Passwords" on page 1132.

Step C. Determine the Authorization Mode for Key Recovery

The Data Recovery Manager allows key recovery agents to authorize recovery of an end user's encryption private key locally or remotely. The default configuration is local authorization. It is important that you evaluate both the authorization modes, and choose the one that is appropriate for your organization. For more information about this, see "Local Versus Remote Key Recovery Authorization" on page 1124.

If want the key recovery agents to authorize key recovery remotely, be sure to set them up as Data Recovery Manager agents following the instructions in "Setting Up Agents" on page 193.

Step D. Customize the Key Recovery Form

Key recovery agents need an appropriate interface to initiate the key recovery process. By default, the Data Recovery Manager's Agent Services interface includes an HTML form (`recoverKey.html`) that allows key recovery agents to initiate the key recovery process and retrieve users' encryption keys; for details about this form, see "Agent Forms and Templates" on page 921.

If you want to customize this form to suit your organization, be careful not to delete any of the information that is vital to the functioning of the form; it is recommended that you restrict your changes to the content presented in the form.

Step E. Configure Key Recovery Policies

This step is optional.

Unlike Certificate Manager and Registration Manager, no policy plug-in modules are provided for the Data Recovery Manager. If you have implemented any custom policies for the Data Recovery Manager's key recovery process, you should make sure that they are configured properly. For details on configuring policies for a subsystem, see "Setting up Policy Rules for a Subsystem" on page 689.

Step 3. Test Your Key Archival and Recovery Setup

The steps outlined in this section explain how to verify key archival and recovery process using Netscape Communicator 4.7 with Personal Security Manager, version 1.01.

Step A. Test Your Key Archival Setup

To test whether you can successfully archive a key, follow these instructions.

1. Enroll for dual certificates

To do this:

1. Open a web browser window.
2. Go to the end-entity interface for the enrollment authority. The default URL is as follows:

   `https://<host_name>:<end_entity_HTTPS_port>` or

   `http://<host_name>:<end_entity_HTTP_port>`
3. In the end-entity interface, open the enrollment form you customized in "Step C. Customize the Certificate Enrollment Form" on page 1136.

   (Perform a "View Source" to ensure that the browser loads the correct page with the JavaScript method you edited.)
4. Fill in all the values and submit the request.

   The client prompts you to enter the password for your key database.
5. When you enter the correct password, the client generates the key pairs.

Do not interrupt the key-generation process.

2. Approve the request

This step is required only if you used the manual enrollment form for requesting the certificate.

1. Go to the enrollment authority's Agent Services interface.

   The default URL is as follows:

   ```
   https://<host_name>:<agent_port>
   ```

2. Click the link that says List Requests.

3. In the form that appears, select the "Show pending requests" option and click Find.

   You should see your request in the list of pending requests.

4. Make sure the request has the appropriate extensions set for S/MIME (E-mail bit of the Netscape Certificate Type extension selected) and the subject name contains the email information (the value of the E attribute).

5. Locate and approve the request.

3. Check if the certificates have been issued

To do this:

1. Click the List Requests link again.

2. In the form that appears, select the "Show completed requests" option and click Find.

   You should see two new certificates with consecutive serial numbers.

3. Download the certificates to the web browser.

4. Go to the security information window of the browser (from the Communicator menu, choose Tools, and then choose Security Info).

5. Click Certificates and then click Mine.

6. Verify that the test certificates have been stored in the browser's certificate database.

You will see only one entry for both the certificates you downloaded. If you select the entry and click View, you should see two certificates—one certificate for encipherment and another one for digital signature.

4. Check if the key has been archived

To do this:

1. Go to the Data Recovery Manager's Agent Services interface.
2. Click the link that says List Requests.
3. In the form that appears, check the "Show completed requests" option and click Find.
4. If the key has been archived successfully, you should see the information pertaining to that key. If you don't see the key archived, check the logs and correct the problem before proceeding to the next step.
5. If the key has been successfully archived, exit the client completely—that is, from the File menu, select Exit; this will close all client windows.

Step B. Verify the Key

To do this:

1. Open a browser window again.
2. Open the email client, Messenger, and send a signed and encrypted email to yourself.
3. When you receive the email, open it, and check the message to see if it is signed and encrypted.

   There should be a security icon at the top-right corner of the message window and it should indicate that the message is signed and encrypted.

Step C. Delete the Certificate

To do this:

1. Open the security information window.
2. Click Certificates and then click Mine.
3. In the list of certificates, select the test certificate you downloaded previously and click Delete.
4. When prompted confirm the delete action.
5. Check your email again. This time, you should not be able to verify the email message because you have deleted the certificates from the client's certificate database.

Step D. Test Your Key Recovery Setup

To test whether you can successfully recover an archived key:

1. Go to the Data Recovery Manager's Agent Services interface.
2. Click the Recover Keys link.
3. In the form that appears, enter any of the following information for the encryption private key that has been archived:

   - The key owner's name
   - The serial number of the key
   - The public key that corresponds to the private key (in the form of base-64 encoded certificate)
   - The instance ID of the enrollment authority that initiated the key archival process

   If you need more information about any of the fields in this form, click the Help button.
4. Click Show Key.

If the key has been archived successfully, you should see the information pertaining to that key.

5. Click Recover.

6. In the form that appears, enter the following information:

   - The PKCS #12 password; the Data Recovery Manager uses this password to encrypt the PKCS #12 package (see "How Agent-Initiated Key Recovery Works" on page 1126).
   - The base-64 encoded certificate that corresponds to the private key you want to recover; use the enrollment authority's end-entity or agent interface to get this information. If you searched for the archived key by providing the base-64 encoded certificate in (step 4), then you don't have to provide this information.
   - The key recovery agents' passwords.

7. Click Recover.

   If you entered the correct information, the Data Recovery Manager returns the private key packaged as a PKCS #12 blob (it contains the recovered key pair and the corresponding certificate) and prompts you to save it. Specify the path and filename for saving the encrypted file. Be sure not to change the default file extension (`.p12`).

Step D. Restore the Key in the Browser's Database

To do this:

1. Go to the security information window of your browser.

2. Import the `*.p12` file (that you saved in the previous step) back into the browser.

3. Open the test email that you couldn't verify after deleting the certificate from the browser's certificate database; you should be able to verify it again.

11

Appendixes

Distinguished Names

Backing Up and Restoring Data

Command-Line Utilities

Certificate Database Tool

Key Database Tool

Netscape Signing Tool

SSL Strength Tool

SSL Debugging Tool

Appendix

A

Distinguished Names

This appendix explains what a distinguished name is and how Netscape Certificate Management System (CMS) uses distinguished names to automatically update certificate information in your corporate LDAP directory.

The appendix has the following sections:

For the most part, the information presented in this appendix is specific to Netscape Directory Server, an LDAP-compliant directory.

What Is a Distinguished Name?

Distinguished names (DNs) are string representations that uniquely identify users, systems, and organizations. In general, DNs are used in LDAP-compliant directories, such as Netscape Directory Server. In Certificate Management System, you use DNs to identify the owner of a certificate and the authority that issued a certificate.

Note If you are using an LDAP directory in conjunction with Certificate Management System, the DNs in your certificates should match the DNs in your directory.

Distinguished Name Components

A DN identifies an entry in an LDAP directory. Because directories are hierarchical, DNs identify the entry by its location as a path in a *hierarchical tree* (much as a path in a file system identifies a file). Generally, a DN begins with a specific common name, and proceeds with increasingly broader areas of identification until the country name is specified. DNs are typically made up of the following components (which are defined in the X.520 standard):

```
CN=common name, OU=organizational unit, O=organization,
L=locality, ST=state or province, C=country name
```

These components are described in Table A.1. For more information on distinguished names, see RFC 2253 (which replaces RFC 1779). You can find RFC 2253 at this URL: `http://www.ietf.org/rfc/rfc2253.txt`

Note that if used in conjunction with an LDAP-compliant directory, Certificate Management System by default recognizes components that are listed in Table A.2.

Table A.1 Definitions of standard DN components

Component	Name	Definition
`CN`	Common name	A required component that identifies the person or object defined by the entry. For example: • `CN=Jane Doe` • `CN=corpDirectory.siroe.com`
`E` (deprecated)	Email address	Identifies the email address of the entry. For example: `jdoe@siroe.com` The use of this component is discouraged by the PKIX standard; instead, it recommends the use of *Subject Alternative Name Extension* to associate an email address with a certificate; see "Subject Alternative Name Extension Policy" on page 668. The reason for this is because it is usually too hard to have a `E` in a directory structure; email addresses change too frequently.

Table A.1 Definitions of standard DN components (Continued)

Component	Name	Definition
OU	Organizational unit	Identifies a unit within the organization. For example: • OU=Sales • OU=Manufacturing
O	Organization	Identifies the organization in which the entry resides. For example: • O=Siroe Corporation • O=Public Power & Gas
L	Locality	Identifies the place where the entry resides. The locality can be a city, county, township, or other geographic region. For example: • L=Mountain View • L=Pacific Northwest • L=Anoka County
ST	State or province name	Identifies the state or province in which the entry resides. For example: • ST=California • ST=British Columbia
C	Country	Identifies the name of the country under which the entry resides. For example: • C=US • C=GB
DC	Domain component	Identifies the domain components of a domain. For example, if the domain is siroe.com, the domain components would be: • DC=siroe, DC=com

Root Distinguished Name

The root distinguished name, or *root DN*, is the first, or top-most, entry in an LDAP directory tree. In Netscape Directory Server, the root DN is commonly referred to as the *directory manager*. By default, the root DN uses no suffix; it is simply a common name attribute-data pair: `CN=Directory Manager`. For example, the root entry's DN could look like this: `CN=Directory Manager, O=Siroe Corporation, C=US`.

Base Distinguished Name

The base distinguished name, or *base DN*, identifies the entry in the directory from which searches initiated by LDAP clients occur; the base DN is often referred to as the search base. For example, if you specify a base DN of `OU=people, O=siroe.com` for a client, the LDAP search operation initiated by the client examines only the `OU=people` subtree in the `O=siroe.com` directory tree.

Typically, an LDAP search consists of the following components:

- The base DN—for example, `O=Siroe`, `C=US`, which initiates a subtree search through all entries below this entry in the directory (in other words, all entries with the suffix `O=Siroe`, `C=US`).
- The search type, which can be a base search (only the entry specified by the base DN is searched), a one-level search (only entries one level below the base entry are searched), or a subtree search (all entries at all levels below the base entry are searched).
- The search filter, which specifies the search criteria applied to each entry within the scope of the search.

When Certificate Management System is configured for LDAP publishing, the search point and search criteria are determined by the configuration parameter values; for details, see information about the mapper or publisher classes in "Modules for Publishing Certificates and CRLs" on page 731. In the absence of a base DN value, Certificate Management System uses DN components in the certificate's subject name to construct the base DN so that it can search the directory in order to publish to or update the appropriate directory entry.

Typically, when you configure Certificate Management System for LDAP publishing, you set the base DN value to `Directory Manager`, so that it can use the publishing directory's root entry to start searching; see "Step 5. Identify the Publishing Directory" on page 830. This way, the entire directory tree is searched.

DNs in Certificate Management System

In Certificate Management System, the characters allowed in a DN are based on the components (attributes) as defined in the X.509 standard.

Table A.2 lists the attributes supported by default and their character sets. Explanation of the character sets are in Table A.3. The set of attributes is extensible.

Table A.2 Allowed characters for value types

Attribute	Value type	Object identifier
CN	Directory String	2.5.4.3
OU	Directory String	2.5.4.11
O	Directory String	2.5.4.10
C	Printable String of length 2	2.5.4.6
L	Directory String	2.5.4.7
ST	Directory String	2.5.4.8
STREET	Directory String	2.5.4.9
TITLE	Directory String	2.5.4.12
UID	Directory String	0.9.2342.19200300.100.1.1
MAIL	IA5String	0.9.2342.19200300.100.1.3
E	IA5String	1.2.840.113549.1.9.1
DC	IA5String	0.9.2342.19200300.100.1.2.25
SERIALNUMBER (for CEP support)	Printable String	2.5.4.5
UNSTRUCTUREDNAME (for CEP support)	IA5String	1.2.840.113549.1.9.2
UNSTRUCTUREDADDRESS (for CEP support)	Printable String	1.2.840.113549.1.9.8

Table A.3 Explanation of character sets for DNs

Value type	Character set allowed
Printable String	A-Z a-z 0-9 space \ ( ) + , - . / : = ?

Table A.3 Explanation of character sets for DNs (Continued)

Value type	Character set allowed
IA5String	Any 7-bit US ASCII character.

Directory String	Any character in format as specified in *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names* (see `http://www.ietf.org/rfc/rfc2253.txt`). Certificate Management System conforms to all of this standard, including support of using hex numbers to escape characters. The special characters are as follows:
	`,`
	`=`
	`+`
	`<`
	`>`
	`#`
	`;`
	They can be escaped by either a backslash (\) before the character or by surrounding the value in double quotes (“ ”). A few examples are shown below:
	• `Siroe Corp. \, Ltd.`
	• `“Siroe Corp. , Ltd”`
	• `“Siroe Corp. \, Ltd”`
	• `Name \, with \= escaped \+ special \< characters \# like \> or \”\\\;`
	• `“Name , with = special + characters < surrounded > by # quotes; ,=+<>#;”`
	• `Name with escaped characters like return \0D or C with Caron \C4\8D or L\4C`
	• `Name with spaces at beginning and end`
	For additional more examples, check the standards.

Extending Attribute Support

By default, Certificate Management System supports attribute identified in Table A.2 on page 1157. You can extend the list of attributes supported by server. The syntax for adding additional X.500Name attributes (or components) is as follows:

```
X500Name.<NEW_ATTRNAME>.oid=<n.n.n.n>
```

```
X500Name.<NEW_ATTRNAME>.class=<string_to_DER_value_converter_
class>
```

Note the following:

- Value converter class converts a string to a ASN.1 value.
- It must implement `netscape.security.x509.AVAValueConverter` interface.

The string-to-value converter class can be one of these:

- `netscape.security.x509.PrintableConverter`—converts a string to a Printable String value. The string must have only printable characters.
- `netscape.security.x509.IA5StringConverter`—converts a string to a IA5String value. The string must have only IA5String characters.
- `netscape.security.x509.DirStrConverter`—converts a string to a Directory (v3) String. The string is expected to be in DirectoryString format according to RFC 2253.
- `netscape.security.x509.GenericValueConverter`—converts a string character by character in the following order, from smaller character sets to broadest character set: Printable, IA5String, BMPString, Universal String.

For example:

```
X500Name.MY_ATTR.oid=1.2.3.4.5.6
```

```
X500Name.MY_ATTR.class=netscape.security.x509.DirStrConverter
```

Adding New or Proprietary Attributes

To add a new or proprietary attribute that's not supported by Certificate Management System by default:

1. Stop the Certificate Manager.
2. Go to this directory: `<server_root>/cert-<instance_id>/config`
3. Open the configuration file, `CMS.cfg`, in a text editor.
4. Add the new attributes to the configuration file.

   For example, if you want to add three proprietary attributes, `MYATTR1` that is a `directoryString`, `MYATTR2` that is a `IA5String`, and `MYATTR3` that is `PrintableString`s, you would add the following lines at the end of the configuration file:

   ```
   X500Name.attr.MYATTR1.oid=1.2.3.4.5.6
   X500Name.attr.MYATTR1.class=netscape.security.x509.DirStrConverter
   X500Name.attr.MYATTR2.oid=11.22.33.44.55.66
   X500Name.attr.MYATTR2.class=netscape.security.x509.IA5StringConverter
   X500Name.attr.MYATTR3.oid=111.222.333.444.555.666
   X500Name.attr.MYATTR3.class=netscape.security.x509.PrintableConverter
   ```

5. Save your changes and close the file.
6. Next, add each new attribute or component (for example, `MYATTR1`, `MYATTR2` and `MYATTR3`) to the enrollment form. For instructions, see "Adding Attributes to an Enrollment Form" on page 1162.
7. Restart the Certificate Manager.
8. Reload the enrollment page and verify your changes; the new attributes that you added should now show up in the form.
9. To verify that the new attributes are in effect, request a certificate using the manual enrollment form.

Be sure to enter values for the new attributes (so that you can verify whether they appear in certificate subject names). For example, you can enter the following values for the new attributes and look for them in the subject name:

```
MYATTR1: a_value
MYATTR2: a.Value
MYATTR3: aValue
CN: John Doe
O: Siroe Corp
```

10. Go to the agent interface and approve your request.

11. When you receive the certificate, check the subject name. The certificate should show the new attribute values in the subject name.

Adding Attributes to an Enrollment Form

The steps below explain how to add an attribute (or component) to the Manual enrollment form:

1. Go to this directory: `<server_root>/cert-<instance_id>/web/ee`

2. Open the `ManUserEnroll.html` file in a text editor.

3. Find the line with the component name that the new component will follow and copy the table row, using the new component name. For the purposes of this instruction, assume that the new component you want to add is `DC` and that it follows component `OU`. Here's how you would add a table row for `DC` (the lines you need to add are shown in bold):

```
<tr>
  <td valign="TOP">
    <div align="RIGHT">
      <font face="PrimaSans BT, Verdana, Arial, Helvetica,
        sans-serif" size="-1">Organization unit: </font>
    </div>
  </td>
  <td valign="TOP">
      <input type="TEXT" name="OU" size="30"
       onchange="formulateDN(this.form, this.form.subject)">
  </td>
```

```
</tr>

<tr>
  <td valign="TOP">
    <div align="RIGHT">
      <font face="PrimaSans BT, Verdana, Arial, Helvetica,
        sans-serif" size="-1">Domain component: </font>
    </div>
  </td>
  <td valign="TOP">
      <input type="TEXT" name="DC" size="30"
       onchange="formulateDN(this.form, this.form.subject)">
  </td>
</tr>
```

4. Save your changes and close the file.
5. Go to this directory: `<server_root>/cert-<instance_id>/web/ee`
6. Open the `cms-funcs.js` file in a text editor.
7. Find the line with `form.OU != null` (or the component that the new component will follow) and add the `if` block. For example, if the new component is `DC` and comes after `OU`, you need to add the lines shown in bold:

```
if (form.OU != null) {
    if (OU.value != '') {
        if (doubleQuotes(OU.value) == true) {
            alert('Double quotes are not allowed in Org Unit
                   field');
            OU.value = '';
            OU.focus();
            return;
        }
        if (distinguishedName.value != '')
            distinguishedName.value += ', ';
            distinguishedName.value += 'OU=' +
            escapeDNComponent(OU.value);
    }
}
```

```
    if (form.DC != null) {
     if (DC.value != '') {
         if (doubleQuotes(DC.value) == true) {
              alert('Double quotes are not allowed in DC
                    field');
             DC.value = '';
             DC.focus();
             return;
         }
         if (distinguishedName.value != '')
             distinguishedName.value += ', ';
             distinguishedName.value += 'DC=' +
             escapeDNComponent(DC.value);
    }
}
```

8. Save your changes and close the file.

9. Reload the Manual enrollment form in the browser and verify your changes.

10. To verify that the Enroll for a certificate using the new attribute value.

Changing the DER Encoding Order

You can also change the DER-encoding order of a DirectoryString. (The reason for allowing this to be configurable is that different clients support different encodings for historical reasons.)

The syntax for changing the DER-encoding order of a DirectoryString is as follows:

```
X500Name.dirStringEncodingOrder=<list_of_encodings_separated_by
_commas>
```

Possible encoding values are as follows:

- `Printable`
- `IA5String`
- `UniversalString`
- `BMPString`

- `UTF8String`

For example, `X500Name.dirEncodingOrder=Printable,BMPString`.

To change the DirectoryString encoding:

1. Stop the Certificate Manager.
2. Go to this directory: `<server_root>/cert-<instance_id>/config`
3. Open the configuration file, `CMS.cfg`, in a text editor.
4. Add the encoding order to the configuration file.

   For example, if you want to specify two encoding values, `PrintableString` and `UniversalString`, and the encoding order is `PrintableString` first and `UniversalString` next, you would add the following line at the end of the configuration file:

   ```
   X500Name.directoryStringEncodingOrder=PrintableString,Univer
   salString
   ```

5. Save your changes and close the file.
6. To verify that the encoding order are in effect, enroll for a certificate using the manual enrollment form. Use `"John_Doe"` for CN.
7. Go to the agent interface and approve your request.
8. When you receive the certificate, use the `dumpasn1` tool to examine the encoding of the certificate; for details on the `dumpasn1` tool, see Appendix C, "Command-Line Utilities."

   The CN component of the subject name should be encoded as a `UniversalString`.
9. Repeat Steps 6 through 8 above, but use `"John Smith` for CN this time.

   The CN component of the subject name should be encoded as a PrintableString.

Role of Distinguished Names in Certificates

In certificates issued by Certificate Management System, DNs are used to identify the entity that owns the certificate. In all cases, if you are using Certificate Management System with a directory, the format of the DNs in your certificates should match the format of the DNs in your directory. It is not necessary that the names match exactly; certificate mapping allows the subject DN in a certificate to be different from the one in the directory. For more information, see "Mapper Modules" on page 732.

DNs in End-Entity Certificates

In end-entity certificates issued by Certificate Management System, DNs are used to identify the end entity that owns the certified key pair. The end entity is one of the following:

- The individual who owns the certified key pair (for personal or client certificates—to form this type of DN, use the `CN` component to specify the user's full name:

  ```
  CN=<user's_full_name>, OU=<user's_division_name>,
  O=<company_name>, C=<country_name>
  ```

  For example:

  ```
  CN=Jane Doe, OU=Human Resources, O=Siroe Corporation, C=US
  ```

- The server that owns the certified key pair (for SSL server certificates)—to form this type of DN, use the `CN` component to specify the server's fully qualified host name in the form `<machine_name>.<your_domain>.<domain>`:

  ```
  CN=<host_name>, OU=<division_name>, O=<company_name>,
  C=<country_name>
  ```

  For example:

  ```
  CN=corpDirectory.siroe.com, OU=Human Resources, O=Siroe
  Corporation, C=US
  ```

When clients such as Netscape Navigator receive a server certificate, they expect the `CN` component of the certificate's subject to match the host name in the URL. If the name in the certificate and the host name of the server do not match, Navigator notifies the user and gives the user the choice of not connecting to the server.

For example, if Navigator goes to the URL `https://corpDirectory.siroe.com` and receives a certificate from the server, it expects the `CN` component of the certificate's subject to be `corpDirectory.siroe.com`. If the `CN` component has a different value (for example, `corpDir.siroe.com`), Navigator notifies the user that the certificate's subject name does not match the host name in the URL.

DNs in CA Certificates

In CA certificates issued by Certificate Management System (for both root and subordinate CAs), DNs are used to identify the authority who owns the certified key pair.

To form this type of distinguished name, use the `CN` component to specify the name of your CA:

```
CN=<CA_name>, O=<company_name>, C=<country_name>
```

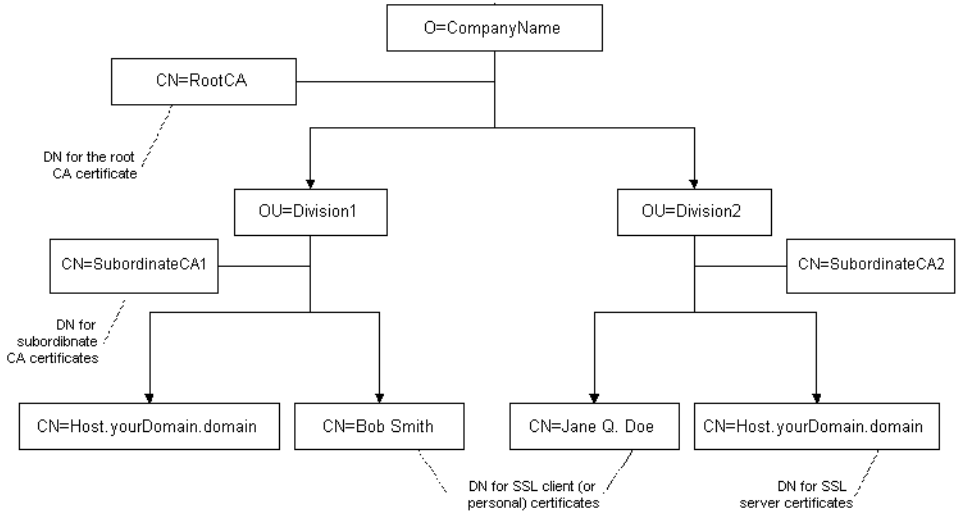
For example:

```
CN=Siroe Certificate Authority, O=Siroe Corporation, C=US
```

Selecting DNs for Certificates

Figure 28.4 illustrates the structure of distinguished names you might select for CA certificates, server certificates, and personal certificates.

Figure 28.4 Sample directory hierarchy



DN Patterns and Certificate Subject Names

You can configure Certificate Management System to issue certificates with subject names that are formulated from the directory attributes and entry DN. The `dnpattern` configuration variable of the automated-enrollment modules, such as `UidPwdDirAuth` and `UidPwdPinDirAuth`, described in "Overview of Authentication Modules" on page 320 enable you to configure the server to issue certificates with required subject names. Note that `dnpattern` is a string representing a subject name pattern to formulate from the directory attributes and entry DN. If empty or not set, Certificate Management System uses the LDAP entry DN as the certificate subject name.

The `dnpattern` configuration variable supports escaped commas and multiple attribute variable assertions (AVAs) in a RDN. Below is the syntax for the DN pattern followed by examples.

Syntax

```
dnPattern := rdnPattern *[ "," rdnPattern ]
rdnPattern := avaPattern *[ "+" avaPattern ]
avaPattern := name "=" value | name "=" "$attr" "." attrName
[ "." attrNumber ] | name "="
"$dn" "." attrName [ "." attrNumber ] | "$dn" "." "$rdn" "."
number
```

Example 1

If the configured DN pattern is

`E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US`

LDAP entry: `dn: UID=jdoe, OU=IS, OU=people, O=siroe.org`

LDAP attributes: `cn: Jane Doe`

LDAP attributes: `mail: jdoe@siroe.org`

The subject name formulated will be as follows:

`E=jdoe@siroe.org, CN=Jane Doe, OU=people, O=siroe.org, C=US`

`E=` the first '`mail`' LDAP attribute value in user's entry.

`CN=` the (first) '`cn`' LDAP attribute value in the user's entry.

`OU=` the second '`ou`' value in the user's entry DN.

`O=` the (first) '`o`' value in the user's entry DN.

`C=` the string 'US'

Example 2

If the configured DN pattern is

`E=$attr.mail.1, CN=$attr.cn, OU=$dn.ou.2, O=$dn.o, C=US`

LDAP entry: `dn: UID=jdoe, OU=IS+OU=people, O=siroe.org`

LDAP attributes: `cn: Jane Doe`

LDAP attributes: `mail: jdoe@siroe.org`

The subject name formulated will be as follows:

`E=jdoe@siroe.org, CN=Jane Doe, OU=people, O=siroe.org, C=US`

`E=` the first '`mail`' LDAP attribute value in user's entry.

`CN=` the (first) '`cn`' LDAP attribute value in the user's entry.

`OU=` the second '`ou`' value in the user's entry DN; note the multiple AVAs in a RDN in this example.

`O=` the (first) '`o`' value in the user's entry DN.

`C=` the string 'US'

Example 3

If the configured DN pattern is

```
CN=$attr.cn, $rdn.2, O=$dn.o, C=US
```

LDAP entry: `dn: UID=jdoe, OU=IS+OU=people, O=siroe.org`

LDAP attributes: `cn: Jane Doe`

LDAP attributes: `mail: jdoe@siroe.org`

The subject name formulated will be as follows:

```
CN=Jane Doe, OU=IS+OU=people, O=siroe.org, C=US
```

`CN=` the (first) '`cn`' LDAP attribute value in the user's entry followed by the second RDN in the user's entry DN.

`O=` the (first) '`o`' value in the user's entry DN.

`C=` the string 'US'

Example 4

If the configured DN pattern is

```
CN=$attr.cn, OU=$dn.ou.2+OU=$dn.ou.1, O=$dn.o, C=US
```

LDAP entry: `dn: UID=jdoe, OU=IS+OU=people, O=siroe, org`

LDAP attributes: `cn: Jane Doe`

LDAP attributes: `mail: jdoe@siroe.org`

The subject name formulated will be as follows:

```
CN=Jane Doe, OU=people+OU=IS, O="siroe \, org", C=US
```

`CN=` the (first) '`cn`' LDAP attribute value in the user's entry.

`OU=` the second '`ou`' value in the user's entry DN followed by the first '`ou`' value in the user's entry; note the multiple AVAs in a RDN in this example.

`O=` the (first) '`o`' value in the user's entry DN.

`C=` the string ‘US’

If an attribute or subject DN component does not exist, the attribute is skipped.

Appendix

B

Backing Up and Restoring Data

This appendix explains how to back up the CMS data and configuration information and how to use the backups to restore data if there is a need.

The appendix has the following sections:

Backup and Restore Tools

Certificate Management System provides tools to backup and restore the data and configuration for a CMS instance. These tools can be used, for example, to back up just your CMS data before you upgrade hardware or software on a machine. You might also use these tools as part of your overall system backup plan, perhaps to provide more frequent checkpoints of the CMS data than a nightly disk backup would record.

Since only CMS configuration and data are backed up, you will need to take other measures to back up data for external PKCS#11 cryptographic or key storage devices (such as smart card readers). If you rely on an external device for key storage (for example, to store the CA signing key), make sure that its

data is backed up whenever you back up CMS data. When you restore the CMS data, it will rely on the external keys still being available. Refer to the PKCS#11 module vendor's instructions for how to back up the data.

The backup and restore tools are simple Perl scripts; most Perl programmers should find no difficulty in customizing or extending them. Read this appendix to familiarize yourself with how the scripts work as well as their capabilities and limitations.

The Perl scripts that perform the backup or restore are called from shell scripts installed in the `<server_root>/cert-<instance_id>/` directory of every CMS instance:

- `cmsbackup[.bat]` copies all of the pertinent data and configuration files for a CMS instance, the local Administration Server, and local Netscape Directory Servers that the instance uses into an compressed archive (a zip file). See "Backing Up Data" on page 1174 for instructions on how to use this tool.
- `cmsrestore[.bat]` opens a named archive, extracts the data, and uses it to restore the configuration of a CMS instance. You have the option to restore everything or to select a subset of the archived data. See "Restoring Data" on page 1179 for instructions on how to use this tool.

Be aware that the backup archives contain sensitive information (for example, your CMS key database). Protect the backup archives as carefully as you protect the server itself. The backups are stored on a local disk by default. To avoid losing both the current data and the backup because of a disk failure, move the backup archives to another medium as soon as they are created. If possible, encrypt the archives or store them on removable media in a secured location.

Backing Up Data

Backing up your data is actually a very simple process. You run the script, and it creates an archive that you store securely. This section explains what the backup tool (`cmsbackup`) does and does not do so that you can plan your overall system maintenance and backup procedures.

What the Backup Tool Does

There is a script or batch file installed in the instance directory of every CMS instance. This file calls the Perl script `<server_root>/bin/cert/tools/CMSBackup.pl` (using a Perl 5.003 interpreter bundled with Certificate Management System). `CMSBackup.pl` does the following:

- Creates a log file where all backup actions are logged
- Creates a temporary backup directory
- Copies non-CMS certificate and key databases and shared files
- Copies files required to configure the Netscape Console and Administration Server
- Backs up the configuration directory server using that server's `db2bak` backup utility (if the server is running locally)
- Backs up the CMS internal database directory server using that server's `db2bak` backup utility
- Copies CMS global and local class files
- Copies CMS user interface files and templates
- Copies CMS instance configuration files
- Creates a compressed archive of all files in the backup directory

The log file is in `<server_root>/cert-<instance_id>/logs/cmsbackup.log`. You should review the log file after each backup to make sure that all phases of the backup completed successfully. If all or part of the backup fails it is usually due to a directory that is missing or not readable by the user running the backup.

The default temporary backup directory is `/var/tmp` (Unix) or `C:\Temp` (Windows NT). Ensure that access to this directory is restricted so that no one can intercept backup files while the archive is being built. You can change the working backup directory by changing the value of `$backup_path_prefix` in `CMSBackup.pl`.

The non-CMS databases and shared files that are backed up are:

- `<server_root>/alias/*`
- `<server_root>/shared/config/*.conf`

The Netscape Console Administration Server files that are backed up are the following files from `<server_root>/admin-serv/config/`:

- `admpw`, the Administration Server password cache
- `*.conf`, the Configuration files for the server and its associated LDAP data
- `*.db`, the certificate and key databases for the Administration Server and `secmodule.db` (database of PKCS#11 modules available to all server instances)

The backup tool will use the Netscape Directory Server `db2bak` tool to create a backup of the CMS server instance internal database directory and the configuration directory (if it is running locally). See Chapter 4, "Managing Directory Server Databases," in *Netscape Directory Server Administrator's Guide* for full details of what this tool does. The data backed up includes all schema and object class definitions and, of course, all entries in the directory.

These CMS global and local class files are Java classes for custom plug-ins used by CMS servers. To back up this data, all files and subdirectories in the following directories are backed up:

- `<server_root>/bin/cert/classes`
- `<server_root>/cert-<instance_id>/classes`

The CMS user interface files and templates are the files used to create the forms end entities and agents use to interact with CMS servers. All of these files for the instance you are backing up are in

- `<server_root>/cert-<instance_id>/web`

The CMS configuration files that get backed up are in `<server_root>/cert-<instance_id>/config`. The specific files and their purposes are:

- `CMS.cfg`, the current master configuration file for the instance.
- `CMS.cfg.*`, previous configuration files, available for reverting to an earlier configuration.

- `*.db`, the server instance key and certificate databases.
- `*.ldif`, ldif-format files that describe objects in the configuration database.
- `pwcache.p12`, the server instance password cache.

All of the data to be backed up is copied to the temporary backup directory. After all of the data has been copied, the script archives the entire backup directory into a compressed archive using `zip` (a copy of `zip` is installed in `<server_root>/bin/cert/tools/zip`). The script deletes the backup directory once the zip archive is created.

What the Backup Tool Does Not Do

The `cmsbackup` script backs up only configuration and data related to a single CMS server instance. You may need to back up other files to recover from a failure completely, depending on the nature of the failure. For example, if some entries in your configuration directory server become corrupted then the data backed up by `cmsbackup` is sufficient to restore the directory to a previous state. If, however, you suffer a catastrophic disk failure, you will probably have to reinstall or restore Certificate Management System, Netscape Console, and Netscape Directory Server binaries and related tools before you use `cmsrestore` to recover your previous configuration.

The following is a list of items which may be part of your overall CMS deployment, but which are not backed up by `cmsbackup`:

- Other instances of CMS servers in the same server root

  Each instance has a copy of the cmsbackup script that backs up only data related to that instance.

- External PKCS#11 module data

  If you use an external PKCS#11 device for key storage, make sure you follow the vendor's instructions for backing up its data whenever you back up your CMS server. It may be possible to extend the `CMSBackup.pl` and `CMSRestore.pl` Perl scripts to include this data in the archives used by the CMS backup tools.

- Server binaries, libraries, and tools

These files do not change after installation, and are not backed up. To restore these files, you can install the software again from the original media. You can also use a more generic disk backup tool to archive the contents of all directories beneath the server root.

Running the Backup Tool

Before you run `cmsbackup`, make sure that

- You are logged in as a user with permission to run `cmsbackup`, to run `db2bak` for the LDAP servers, and to write to the output directory; you may need to become superuser on a UNIX system or Administrator on a Windows NT system.
- There is plenty of disk space in the output directory; the size of the backup archive will vary with the amount of data in your system, so you will learn from experience how much space you require.

The configuration that you back up, of course, will use all of your current passwords. You will need to remember the current passwords if you restore this data after you change some passwords.

To run `cmsbackup`:

1. Log in to the machine where your CMS instance is running and open a command shell.
2. Change to the CMS server instance directory in the server root. For example, if your server root is `/usr/netscape/server4` and the instance id of the server you want to back up is `cmsinstance`:

   ```
   # cd /usr/netscape/server4/cert-cmsinstance
   ```

3. Execute the backup script: either `cmsbackup` on UNIX or `cmsbackup.bat` on Windows NT systems. For example,

   ```
   # ./cmsbackup
   ```

The script will run. Control returns to the command prompt when the script has finished.

After You Finish a Backup

Immediately after running the backup tool, you should check the log file to make sure that all systems were archived successfully. The log file is

```
<server_root>/cert-<instance>/logs/cmsbackup.log
```

If the any part of the backup was not successful, there will be a message labeled `WARNING` or `ERROR` that tells you why. Most of the time, the problems are the result of directories or files that are missing or inaccessible to the user running `cmsbackup`. If necessary, change the permissions on the required files, delete the zip archive in the output directory, and run `cmsbackup` again.

Once you have a successful zip archive, you should secure it. The output directory is probably accessible to any user on the system, and it may be on the same physical disk as the server instance itself. You want to make sure the archive is not accessible to unauthorized users and that you can use the archive if there is a system hardware failure. Remember, the archive contains a database of private keys. Although it is not easy to extract a key from the database without the correct passwords, you do not want anyone to have the opportunity to try.

Move the zip archive to another machine or removable medium. If possible, encrypt the archive (do not use the private keys stored in your CMS server's database, since they may not be available when you need to restore the data). If you copy the archive to removable media such as tape or CD, make sure the copy is kept in a limited-access, locked area.

Restoring Data

The purpose of creating back up archives, of course, is to allow you to restore a previous state of the CMS server instance after a hardware or software failure corrupts your current state. The restore tool allows you to recover all or part of the configuration that was backed up. For example, you can use the tool to restore just the internal database of a CMS server instance.

A special case, automatic restore, allows you to completely restore the configuration from the latest backup archive quickly and without interaction.

Before You Restore Data

Before you can restore from a backup archive, the archive you want to use has to be available on a disk accessible from the server instance directory. If you want to use the automatic restore feature, you should put the archive in the output directory where `cmsbackup` originally created it (`C:\Temp` on Windows NT or `/var/tmp` on UNIX).

Note the full path name to the backup archive; in the instructions later it will be referred to as `<archive_path>`. For example, on a UNIX system, the `<archive_path>` might be

`/var/tmp/CMS_cmsdemo_BACKUP-19991115093827.zip`.

You can use the word `automatic` instead of a path name to indicate the location of the backup archive. If you use `automatic`, the restore tool will read the file `logs/latest_backup` to find the path name of the archive. This file is created by `cmsbackup` and contains the name of the last archive created. Note that `automatic` always causes *all* data to be restored: you will not be able to select only a subset of the data.

If you moved the zip archive to another machine or removable medium, copy it back to the local file system. If you encrypted the archive, decrypt it before you try to restore the data.

The user running the restore tool will probably need superuser (UNIX) or Administrator (Windows NT) privileges. The user running the tool will need privileges to do the following:

- Read the backup zip archive
- Create a temporary working directory in the directory where the archive is located
- Create directories and files in the server root and server instance directories (for example, if the `CMS.cfg` file needs to be restored)
- Run the `bak2db` tool for any Netscape Directory Servers that are being restored
- (UNIX) Change file ownership of the LDAP database backup files to the directory server user. The directory server user is defined by the `localuser` parameter in `slapd.conf`. If the directory server user is

different from the user running `cmsrestore`, the user running the tool must be able to run `chown` to change the owner of the files to the LDAP server user (typically only the superuser has this privilege).

The process of restoring data will require that some servers be stopped and restarted. If any of your servers require passwords to start (for example, if they need to unlock the key database in order to listen for SSL requests), you will be prompted for the password. If any passwords have changed since you created the backup archive, make sure you know the password that was valid at the time the archive was created.

Running the Restore Tool

To run `cmsrestore`:

1. Log in to the machine where the CMS instance you want to restore is installed and open a command shell.

2. Change to the CMS server instance directory in the server root. For example, if your server root is `/usr/netscape/server4` and the instance id of the server you want to restore is `cmsinstance`:

   ```
   # cd /usr/netscape/server4/cert-cmsinstance
   ```

3. Execute the restore script: either `cmsrestore` on UNIX or `cmsrestore.bat` on Windows NT systems.

   You can either provide the `<archive_path>` as an argument or use the argument `automatic` (to read the archive path from `logs/latest_backup`):

   ```
   # ./cmsrestore <archive_path> | automatic
   ```

   For example,

   ```
   # ./cmsrestore \
   /var/tmp/CMS_cmsdemo_BACKUP-19991115093827.zip
   ```

   If you use `automatic` as the argument, the restore proceeds automatically; go to Step 9 when `cmsrestore` completes.

4. The script asks if you would like to perform a complete or prompted restore. Enter

   - `c` (complete) to completely restore the contents of the archive without further prompts. Proceed with Step 9 when the restore is complete.
   - `p` (prompted) to have the script ask you whether you want to restore specific parts of the archive.

5. If the configuration directory server is located in the same server root, the first prompt asks if you want to restore it. The configuration directory server is the directory used by the Administration Server to store information about servers, users, and groups.

   If you answer `yes`, the restore tool stops the directory server, restores the data, then restarts the server. You may be asked to enter a password if one is required to start the server.

6. Next you are asked if you want to restore selected Administration Server data.

   If you answer `no`, no Administration Server data will be restored; proceed with the next step.

   If you answer `yes`, you will be asked three more questions about specific Administration Server data you want to restore:

   1. Main admin data is data in the Administration Server's config directory.
   2. Non-CMS shared data is data in the `<server_root>/shared/config` directory.
   3. Non-CMS certificate and key databases are the databases in the `<server_root>/alias` directory; CMS instances maintain their own `alias` directories in the instance subdirectories.

   After you answer the questions, the Administration Server is stopped, the data restored from the archive, and the server is started again. If necessary, you will be prompted to enter a password to start the Administration Server.

7. Next you are asked if you want to restore the CMS internal database directory server. This is the directory server this CMS instance uses as its internal database.

If you answer `yes`, the restore tool stops the directory server, restores the data, then restarts the server. You may be asked to enter a password if one is required to start the server.

8. Next you are asked if you want to restore selected data for this CMS server instance.

   If you answer `yes`, you will be asked four more questions about the following CMS server instance data that you can restore:

   1. Global CMS classes are Java classes that are shared by all CMS servers in the same server root.

   2. Critical CMS data includes the configuration files, certificate and key databases, and password cache in the `config` directory for this CMS instance.

   3. Local CMS classes are Java classes used only by this server instance.

   4. Custom CMS UI data includes all HTML files and templates in the web subdirectory of this CMS instance.

   After you answer these questions, the tool stops the CMS server, restores the data, then restarts the server. You will be asked to enter the single sign-on password that unlocks the password cache when the server restarts; see "Password Cache" on page 144.

9. After the tool finishes restoring data, view the `cmsrestore.log` file in the server instance `logs` directory.

   Review each step to make sure there were no errors in restoring the data. If there were errors or warnings, you may want to run cmsrestore again. You may need to change permissions on some files or manually start some servers before running `cmsrestore` again.

The restore tool deletes the working directory where it unpacked the archive, but it does not delete the archive itself. You probably will not want to keep the backup archive on disk. Remember that the backup archive contains sensitive information. Delete or secure the archive when you are done using it to restore data.

Appendix

C

Command-Line Utilities

Netscape Certificate Management System (CMS) is bundled with various command-line utilities. This appendix summarizes these utilities, explains a few of them, and provides pointers for the rest.

The appendix has the following sections:

Summary of Command-Line Utilities

Table C.1 summarizes the command-line utilities that are bundled with Certificate Management System.

Table C.1 Summary of command-line utilities

Utility/Tool	Function
Batch/Shell Scripts located under <server_root>/bin/cert/tools/ (require jre):	
`AtoB` (ASCII to Binary Tool)	Converts ASCII base-64 encoded data to binary base-64 encoded data. For details, see "ASCII to Binary Tool" on page 1189.
`BtoA` (Binary to ASCII Tool)	Converts binary base-64 encoded data to ASCII base-64 encoded data. For details, see "Binary to ASCII Tool" on page 1189.
`PasswordCache` (Password Cache Utility)	Manipulates the contents of the single sign-on password cache. For details, see "Password Cache Utility" on page 146.
`PrettyPrintCert` (Pretty Print Certificate Tool)	Prints the contents of a certificate stored as ASCII base-64 encoded data in a human-readable form. For details, see "Pretty Print Certificate Tool" on page 1190.
`PrettyPrintCrl` (Pretty Print CRL Tool)	Prints the contents of a CRL stored as ASCII base-64 encoded data in a human-readable form. For details, see "Pretty Print CRL Tool" on page 1193.
Perl Scripts located under <server_root> (require_perl):	
`cmsbackup`	Copies all of the pertinent data and configuration files for a CMS instance, the local Administration Server, and local Netscape Directory Servers that the instance uses into a compressed archive. For details, see "Backing Up Data" on page 1174.
`cmsrestore`	Opens a named archive, extracts the data, and uses it to restore the configuration of a CMS instance. For details, see "Restoring Data" on page 1179.
Executable tools located under <server_root>/shared/bin:	
`modutil` (Security Module Database Tool)	Used for managing the PKCS #11 module information within `secmod.db` files or within hardware tokens. For details, see "modutil" in Appendix B of *Managing Servers with Netscape Console*. To locate this document, open the `<server_root>/manual/index.html` file.
Executable tools located under <server_root>/bin/cert/tools:	
`certutil` (Certificate Database Tool)	View and manipulate the certificate database (`cert7.db`) contents. For details, see Appendix D, "Certificate Database Tool" in the online version of this document.

Table C.1 Summary of command-line utilities (Continued)

Utility/Tool	Function
`keyutil` (Key Database Tool)	View and manipulate the key database (`key3.db`) contents. For details, see Appendix E, "Key Database Tool" in the online version of this document.
`killproc`	Kills or terminates system processes in Windows NT. For details, see "Attending to an Unresponsive Server" on page 142.

`migrate/AIX/migrate`
`migrate/HP-UX/migrate`
`migrate/Solaris/migrate`
`migrate/WINNT/migrate`
(Migration Tool)
	Migrates data from a Certificate Server 1.x installation into a Certificate Management System installation. For details, see "Appendix A, Migrating from Certificate Server" in the *Netscape Certificate Management System Installation and Deployment Guide*.

`setpin` (PIN Generator tool)	Generates PINs for end users for directory- and PIN-based authentication. For details, see "Using the PIN Generator Tool" on page 369.
`signtool` (Netscape Signing Tool)	Digitally signs any file, including log files. For details, see Appendix F, "Netscape Signing Tool" in the online version of this document.
`sslstrength` (SSL Strength Tool)	Connects to an SSL server and reports back the type and strength of the encryption cipher that it's using. For details, see Appendix G, "SSL Strength Tool" in the online version of this document.
`ssltap` (SSL Debugging Tool)	Used to debug SSL applications. For details, see Appendix H, "SSL Debugging Tool" in the online version of this document.
Third-party executable tools located under <server_root>/bin/cert/tools:	
`dumpasn1`	Dumps the contents of binary base-64-encoded data. For details, see "dumpasn1 Tool" on page 1195.
Third-party support tools located under <server_root>:	
`bin/base/jre/bin/jre` `bin/cert/jre/bin/jre`	Java runtime executable for Netscape Console. Java runtime executable for Certificate Management System. Note: The CMS jre is invoked as `cms_daemon` during CMS installation and configuration, as `cms_watchdog` to monitor the status of the CMS server, and as `cms_server` to actually run the CMS server.
`bin/cert/tools/unzip`	Decompression utility executable.

Table C.1 Summary of command-line utilities (Continued)

Utility/Tool	Function
`bin/cert/tools/zip`	Compression utility executable.
`install/perl`	`perl` scripting language executable.

The `AtoB`, `BtoA`, `PrettyPrintCert`, `PrettyPrintCrl`, and `dumpasn1` tools are useful for converting back and forth between various encodings and formats you may encounter when dealing with keys and certificates. (These tools are explained in this appendix.)

The Password Cache Utility can be used to manipulate the contents of an existing single sign-on password cache and to create a new cache.

The Certificate Database Tool, Key Database Tool, and Security Module Database Tool are useful for a variety of administrative tasks that involve manipulating certificate and key databases. Note that the The Security Module Database Tool is explained in section entitled "modutil" in Appendix B of *Managing Servers with Netscape Console*. To locate this document, open the `<server_root>/manual/index.html` file.

The Migration tool is used to convert Certificate Server 1.x data for use with Certificate Management System, and the PIN Generator tool is used to create PINs for directory authentication. The `killproc` tool is used to terminate the Java virtual machines, called `jre` processes, when Certificate Management System becomes unresponsive.

The Netscape Signing Tool can be used to associate a digital signature with any file, including CMS log files.

The SSL Strength Tool and SSL Debugging Tool are useful for testing and debugging purposes.

ASCII to Binary Tool

You can use the ASCII to Binary tool to convert ASCII base-64 encoded data to binary base-64 encoded data.

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run the ASCII to Binary tool, type the following command:

```
AtoB[.bat] <input_file> <output_file>
```

`.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

`<input_file>` specifies the path to the file that contains the base-64 encoded data in ASCII format.

`<output_file>` specifies the path to the file to write the base-64 encoded data in binary format.

Example

```
AtoB.bat C:\test\data.in C:\test\data.out
```

The above command takes the base-64 encoded data (in ASCII format) in the file named `data.in` and writes the binary equivalent of the data to the file named `data.out`.

Binary to ASCII Tool

You can use the Binary to ASCII tool to convert binary base-64 encoded data to ASCII base-64 encoded data.

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run the Binary to ASCII tool, type the following command:

```
BtoA[.bat] <input-file> <output_file>
```

`.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

`<input_file>` specifies the path to the file that contains the base-64 encoded data in binary format.

`<output_file>` specifies the path to the file to write the base-64 encoded data in ASCII format.

Example

```
BtoA.bat C:\test\data.in C:\test\data.out
```

The above command takes the base-64 encoded data (in binary format) in the file named `data.in` and writes the ASCII equivalent of the data to the file named `data.out`.

Pretty Print Certificate Tool

You can use the Pretty Print Certificate tool to print the contents of a certificate stored as ASCII base-64 encoded data in a human-readable form.

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run the Pretty Print Certificate tool, type the following command:

```
PrettyPrintCert[.bat] <input_file> [<output_file>]
```

`.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

`<input_file>` specifies the path to the file that contains the base-64 encoded certificate.

`<output_file>` specifies the path to the file to write the certificate. This argument is optional; if you don't specify an output file, the certificate information is written to the standard output.

Example

```
PrettyPrintCert.bat C:\test\cert.in C:\test\cert.out
```

The above command takes the base-64 encoded certificate in the `cert.in` file and writes the certificate in the pretty-print form to the output file named `cert.out`.

The base-64 encoded certificate (content of the `cert.in` file) would look similar to this:

```
-----BEGIN CERTIFICATE-----

MIIC2DCCAkGgAwIBAgICEAwwDQYJKoZIhvcNAQEFBQAwfDELMAkGA1UEBhMCVVMxIzAhBgNVBAoTG
lBhbG9va2FWaWxsZSBXaWRnZXRzLCBJbmMuMR0wGwYDVQQLExRXaWRnZXQgTWFrZXJzICdSJyBVcz
EpMCcGA1UEAxMgVGVzdCBUZXN0IFRlc3QgVGVzdCBUZXN0IFRlc3QgQ0EwHhcNOTkwMjE4MDM0MzM
5WhcNMDAwMjE4MDM0MzM5WjCBrjELMAkGA1UEBhMCVVMxJjAkBgNVBAoTHU5ldHNjYXBlIENvbW11
bmljYXRpb25zIENvcnAuMRUwEwYDVQQLEwOZXRzY2FwZSBDTVMxGDAWBEBEwhtaGFybXNlbjEfMB0
GA1UEAxMWaW50ZGV2Y2EgQWRtaW5pcwp0frfJOObeiSsia3BuifRHBNw95ZZQR9NIXr1x5bEdYM1n
0nksKdflcQJ6mcA7718OZIRMfLKyRaHua24zAAMWjsH4F250gAPfZuiaTUYcBx8rhIvCwsac1Xb4X
zPp1DZO8NX+9A6Zod0CAwEAAaM2MDQwEQYJYIZIAYb4QgEBBAQDAgCgMB8GA1UdIwQYaAFOu1EY8A
mhqmblKUqXS8Zc8HiSojMA0GCgVIHT2xU+055U8omp0kjwHqDkegWhUtfMfeCdbNiXOpwSjCVIf1F
ZvjrML/rCkV9pkn7574EBdaP

-----END CERTIFICATE-----
```

The certificate in pretty-print form (content of the `cert.out` file) would look similar to this:

```
Certificate:

        Data:

            Version:  v3

            Serial Number: 0x100C

            Signature Algorithm: OID.1.2.840.113549.1.1.5 -
1.2.840.113549.1.1.5

            Issuer: CN=Test Test CA,OU=Widget Makers 'R'Us,O=PalookaVille
Widgets\, Inc.,C=US
```

```
            Validity:

                Not Before: Wednesday, February 17, 1999 7:43:39 PM

                Not  After: Thursday, February 17, 2000 7:43:39 PM

            Subject: MAIL=admin@netscape.com,CN=testCA,Administrator
UID=admin,OU=Netscape CMS,O=Netscape Comm Corp.,C=US

            Subject Public Key Info:

                Algorithm: RSA - 1.2.840.113549.1.1.1

                Public Key:

                    30:81:89:02:81:81:00:DE:26:B3:C2:9D:3F:7F:FA:DF:

                    24:E3:9B:7A:24:AC:89:AD:C1:BA:27:D1:1C:13:70:F7:

                    96:59:41:1F:4D:21:7A:F5:C7:96:C4:75:83:35:9F:49:

                    E4:B0:A7:5F:95:C4:09:EA:67:00:EF:BD:7C:39:92:11:

                    31:F2:CA:C9:16:87:B9:AD:B8:39:69:18:CE:29:81:5F:

                    F3:4D:97:B9:DF:B7:60:B3:00:03:16:8E:C1:F8:17:6E:

                    7A:D2:00:0F:7D:9B:A2:69:35:18:70:1C:7C:AE:12:2F:

                    0B:0F:EC:69:CD:57:6F:85:F3:3E:9D:43:64:EF:0D:5F:

                    EF:40:FF:A6:68:FD:DD:02:03:01:00:01:

            Extensions:

                Identifier: 2.16.840.1.113730.1.1

                    Critical: no

                    Value:

                        03:02:00:A0:

                Identifier: Authority Key Identifier - 2.5.29.35

                    Critical: no

                    Key Identifier:

                        EB:B5:11:8F:00:9A:1A:A6:6E:52:94:A9:74:BC:65:CF:

                        07:89:2A:23:

        Signature:

            Algorithm: OID.1.2.840.113549.1.1.5 - 1.2.840.113549.1.1.5

            Signature:

                3E:8A:A9:9B:D1:71:EE:37:0D:1F:A0:C1:00:17:53:26:

                6F:EE:28:15:20:74:F6:C5:4F:B4:E7:95:3C:A2:6A:74:

                92:3C:07:A8:39:12:1B:7E:C4:C7:AE:79:C8:D8:FF:1F:

                D5:48:D8:2E:DD:87:88:69:D5:3A:06:CA:CA:9C:9A:55:

                DA:A9:E8:BF:36:BC:68:6D:1F:2B:1C:26:62:7C:75:27:

                E2:8D:24:4A:14:9C:92:C6:F0:7A:05:A1:52:D7:CC:7D:
```

```
E0:9D:6C:D8:97:3A:9C:12:8C:25:48:7F:51:59:BE:3C:
2B:30:BF:EB:0A:45:7D:A6:49:FB:E7:BE:04:05:D6:8F:
```

Pretty Print CRL Tool

You can use the Pretty Print CRL tool to print the contents of a CRL stored as ASCII base-64-encoded data in a human-readable form.

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run the Pretty Print CRL tool, type the following command:

```
PrettyPrintCrl[.bat] <input_file> [<output-file>]
```

`.bat` specifies the file extension; this is required only when running the utility on a Windows NT system.

`<input_file>` specifies the path to the file that contains the base-64 encoded CRL.

`<output_file>` specifies the path to the file to write the CRL. This argument is optional; if you don't specify an output file, the CRL information is written to the standard output.

Example

```
PrettyPrintCrl.bat C:\test\crl.in C:\test\crl.out
```

The above command takes the base-64 encoded CRL in the `crl.in` file and writes the CRL in the pretty-print form to the output file named `crl.out`.

The base-64 encoded CRL (content of the `crl.in` file) would look similar to this:

```
-----BEGIN CRL-----

MIIBkjCBAIBATANBgkqhkiG9w0BAQQFADAsMREwDwYDVQQKEwhOZXRzY2FwZTEXMBUGA1UEAxMOQ2
VydDQwIFRlc3QgQ0EXDTk4MTIxNzIyMzcyNFowgaowIAIBExcNOTgxMjE1MTMxODMyWjAMMAoGA1U
dFQQDCgEBMCACARIXDTk4MTIxNTEzMjA0MlowDDAKBgNVHRUEAwoBAjAgAgERFw05ODEyMTYxMjUx
NTRaMAwwCgYDVR0VBAMKAQEwIAIBEBcNOTgxMjE3MTAzNzI0WjAMMAoGA1UdFQQDCgEDMCACAQoXD
Tk4MTEyNTEzMTExOFowDDAKBgNVHRUEAwoBATANBgkqhkiG9w0BAQQFAAOBgQBCN85O0GPTnHfImY
PROvoorx7HyFz2ZsuKsVblTcemsX0NL7DtOa+MyY0pPrkXgm157JrkxEJ7GBOeogbAS6iFbmeSqPH
j8+JBH5stJNnfTCuhaM6Wx63Wc9LwZXOXTPsvpGxq0YYI0+DPfBZlI3z4lCsNczxJV+9NkeMrheEg
==

-----END CRL-----
```

The CRL in pretty-print form (content of the `crl.out` file) would look similar to this:

```
Certificate Revocation List:
        Data:
            Version:  v2
            Signature Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
            Issuer: CN=Cert40 Test CA,O=Netscape
            This Update: Thu Dec 17 14:37:24 PST 1998
            Revoked Certificates:
                Serial Number: 0x13
                Revocation Date: Tuesday, December 15, 1998 5:18:32 AM
                Extensions:
                    Identifier: Revocation Reason - 2.5.29.21
                        Critical: no
                        Reason: Key_Compromise
                Serial Number: 0x12
                Revocation Date: Tuesday, December 15, 1998 5:20:42 AM
                Extensions:
                    Identifier: Revocation Reason - 2.5.29.21
                        Critical: no
                        Reason: CA_Compromise
                Serial Number: 0x11
                Revocation Date: Wednesday, December 16, 1998 4:51:54 AM
                Extensions:
                    Identifier: Revocation Reason - 2.5.29.21
                        Critical: no
                        Reason: Key_Compromise
                Serial Number: 0x10
                Revocation Date: Thursday, December 17, 1998 2:37:24 AM
```

```
        Extensions:
            Identifier: Revocation Reason - 2.5.29.21
                Critical: no
                Reason: Affiliation_Changed
        Serial Number: 0xA
        Revocation Date: Wednesday, November 25, 1998 5:11:18 AM
        Extensions:
            Identifier: Revocation Reason - 2.5.29.21
                Critical: no
                Reason: Key_Compromise
Signature:
    Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
    Signature:
        42:37:CE:4E:D0:63:D3:9C:77:C8:99:83:D1:3A:FA:28:
        AF:1E:C7:C8:5C:F6:66:CB:8A:B1:56:E5:4D:C7:A6:B1:
        7D:0D:2F:B0:ED:39:AF:8C:C9:8D:29:3E:B9:17:82:6D:
        79:EC:9A:E4:C4:42:7B:18:13:9E:A2:06:C0:4B:A8:85:
        6E:67:92:A8:F1:E3:F3:E2:41:1F:9B:2D:24:D9:DF:4C:
        2B:A1:68:CE:96:C7:AF:F7:5B:F7:3D:2F:06:57:39:74:
        CF:B2:FA:46:C6:AD:18:60:8D:3E:0C:F7:C1:66:52:37:
        CF:89:42:B0:D7:33:C4:95:7E:F4:D9:1E:32:B8:5E:12:
```

dumpasn1 Tool

The `dumpasn1` tool is freeware that is packaged with Certificate Management System for your convenience. You can use this tool to dump the contents of binary base-64 encoded data. For more information about this tool, see this URL:

`http://www.cs.auckland.ac.nz/~pgut001/`

Appendix

D

Certificate Database Tool

Certificate Database Tool is a command-line utility that can create the certificate database file (`cert7.db`) for Certificate Management System. The utility can also list, generate, modify, or delete certificates within the file.

Certificate database management tasks are part of a process that typically also involves managing key databases (`key3.db` files). The key and certificate management process generally begins with creating keys in the key database, then generating and managing certificates in the certificate database.

This appendix discusses certificate database management:

For information on key database and security module database management, see "Key Database Tool" on page 1211 and "modutil" in Appendix B of *Managing Servers with Netscape Console*.

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run Certificate Database Tool, type the following command:

`certutil` *option* [*arguments*]

where *options* and *arguments* are combinations of the options and arguments listed in the following section. Each command takes one option. Each option may take zero or more arguments. To see a usage string, issue the command without options, or with the `-H` option.

Options and Arguments

Options specify an action and are uppercase. Option arguments modify an action and are lowercase. Certificate Database Tool command options and their arguments are defined as follows:

Options	
`-N`	Create a new certificate database.
`-S`	Create an individual certificate and add it to a certificate database.
`-R`	Create a certificate-request file that can be submitted to a certificate authority (CA) for processing into a finished certificate. Output defaults to standard out unless you use `-o` *output-file* argument. Use the `-a` argument to specify ASCII output.

`-C`	Create a new binary certificate file from a binary certificate-request file. Use the `-i` argument to specify the certificate-request file. If this argument is not used Certificate Database Tool prompts for a filename.
`-A`	Add an existing certificate to a certificate database. The certificate database should already exist; if one is not present, this option will initialize one by default.
`-L`	List all the certificates, or display information about a named certificate, in a certificate database. Use the `-h` *tokenname* argument to specify the certificate database on a particular hardware or software token.
`-V`	Check the validity of a certificate and its attributes.
`-M`	Modify a certificate's trust attributes using the values of the `-t` argument.
`-H`	Display a list of the options and arguments used by Certificate Database Tool.

Arguments

`-a`	Use ASCII format or allow the use of ASCII format for input or output. This formatting follows RFC #1113. For certificate requests, ASCII output defaults to standard output unless redirected.
`-b` *validity-time*	Specify a time at which a certificate is required to be valid. Use when checking certificate validity with the `-V` option. The format of the *validity-time* argument is `"YYMMDDHHMMSS[+HHMM|-HHMM|Z]"`. Specifying seconds (`SS`) is optional. When specifying an explicit time, use `"YYMMDDHHMMSSZ".` When specifying an offset time, use `"YYMMDDHHMMSS+HHMM"` or `"YYMMDDHHMMSS-HHMM".` If this option is not used, the validity check defaults to the current system time.

`-c` *issuer*	Identify the certificate of the CA from which a new certificate will derive its authenticity. Use the exact nickname or alias of the CA certificate, or use the CA's email address. Bracket the *issuer* string with quotation marks if it contains spaces.
`-d` *certdir*	Specify a directory containing a certificate database file. On Unix Certificate Database Tool defaults to `$HOME/.netscape` (that is, `~/.netscape`). On Windows NT the default is the current directory. The `cert7.db` and `key3.db` database files must reside in the same directory.
`-e`	Check a certificate's signature during the process of validating a certificate.
`-f` *password-file*	Specify a file that will automatically supply the password to include in a certificate or to access a certificate database. This is a plain-text file containing one password. Be sure to prevent unauthorized access to this file.
`-h` *tokenname*	Specify the name of a token to use or act on. Unless specified otherwise the default token is an internal slot (specifically, internal slot 2). This slot can also be explicitly named with the string `"internal"`. An internal slots is a virtual slot maintained in software, rather than a hardware device. Internal slot 2 is used by key and certificate services. Internal slot 1 is used by cryptographic services.
`-i` *cert*|*cert-request-file*	Specify a specific certificate, or a certificate-request file.
`-k` *shortkeyID*	Specify the public key to use when creating a certificate or certificate request. The *shortkeyID* is the first few bytes of the keyID (as shown by the `keyutil -L` command), starting from the second byte, with a length sufficient to identify it uniquely.
`-l`	Display detailed information when validating a certificate with the `-V` option.

`-m` *serial-number*	Assign a unique serial number to a certificate being created. This operation should be performed by a CA. The default serial number is 0 (zero). Serial numbers are limited to integers.
`-n` *certname*	Specify the nickname of a certificate to list, create, add to a database, modify, or validate. Bracket the *certname* string with quotation marks if it contains spaces.
`-o` *output-file*	Specify the output file name for new certificates or binary certificate requests. Bracket the *output-file* string with quotation marks if it contains spaces. If this argument is not used the output destination defaults to standard output.
`-p` *phone*	Specify a contact telephone number to include in new certificates or certificate requests. Bracket this string with quotation marks if it contains spaces.
`-r`	Display a certificate's binary DER encoding when listing information about that certificate with the `-L` option.
`-s` *subject*	Identify a particular certificate owner for new certificates or certificate requests. Bracket this string with quotation marks if it contains spaces. The subject identification format follows RFC #1485.

`-t` *trustargs*	Specify the trust attributes to modify in an existing certificate or to apply to a certificate when creating it or adding it to a database. There are three available trust categories for each certificate, expressed in this order: "*SSL*, *email*, *object signing*". In each category position use zero or more of the following attribute codes: `p` Valid peer `P` Trusted peer (implies `p`) `c` Valid CA `T` Trusted CA to issue client certificates (implies `c`) `C` Trusted CA to issue server certificates (SSL only) (implies `c`) `u` Certificate can be used for authentication or signing `w` Send warning (use with other attributes to include a warning when the certificate is used in that context) The attribute codes for the categories are separated by commas, and the entire set of attributes enclosed by quotation marks. For example: `-t "TCu,Cu,Tuw"` Use the `-L` option to see a list of the current certificates and trust attributes in a certificate database.
`-u` *certusage*	Specify a usage context to apply when validating a certificate with the `-V` option. The contexts are the following: `C` (as an SSL client) `V` (as an SSL server) `S` (as an email signer) `R` (as an email recipient)

`-v` *valid-months*	Set the number of months a new certificate will be valid. The validity period begins at the current system time unless an offset is added or subtracted with the `-w` option. If this argument is not used, the default validity period is three months. When this argument is used, the default three-month period is automatically added to any value given in the *valid-month* argument. For example, using this option to set a value of `3` would cause 3 to be added to the three-month default, creating a validity period of six months. You can use negative values to reduce the default period. For example, setting a value of `-2` would subtract 2 from the default and create a validity period of one month.
`-w` *offset-months*	Set an offset from the current system time, in months, for the beginning of a certificate's validity period. Use when creating the certificate or adding it to a database. Express the offset in integers, using a minus sign (`-`) to indicate a negative offset. If this argument is not used, the validity period begins at the current system time. The length of the validity period is set with the `-v` argument.
`-x`	Use Certificate Database Tool to generate the signature for a certificate being created or added to a database, rather than obtaining a signature from a separate CA.
`-y rsa|dsa`	Specify the type of key used to generate a new certificate, either RSA or DSA. The default is `rsa`.
`-1`	Add a key usage extension to a certificate that is being created or added to a database. This extension allows a certificate's key to be dedicated to supporting specific operations such as SSL server or object signing. Certificate Database Tool will prompt you to select a particular usage for the certificate's key. These usages are described under "Standard X.509 v3 Certifiate Extensions' in Appendix C of *Netscape Certificate Management System Installation and Deployment Guide*.

`-2`	Add a basic constraint extension to a certificate that is being created or added to a database. This extension supports the certificate chain verification process. Certificate Database Tool will prompt you to select the certificate constraint extension. Constraint extensions are described in "Standard X.509 v3 Certifiate Extensions' in Appendix C of *Netscape Certificate Management System Installation and Deployment Guide*.
`-3`	Add an authority key ID extension to a certificate that is being created or added to a database. This extension supports the identification of a particular certificate, from among multiple certificates associated with one subject name, as the correct issuer of a certificate. Certificate Database Tool will prompt you to select the authority key ID extension. Authority key ID extensions are described under "Standard X.509 v3 Certifiate Extensions' in Appendix C of *Netscape Certificate Management System Installation and Deployment Guide*.
`-4`	Add a CRL distribution point extension to a certificate that is being created or added to a database. This extension identifies the URL of a certificate's associated certificate revocation list (CRL). Certificate Database Tool prompts you to enter the URL.

Usage

Certificate Database Tool's capabilities are grouped as follows, using these combinations of options and arguments. Options and arguments in square brackets are optional, those without square brackets are required.

- Creating a new `cert7.db` file:

  `-N [-d` *certdir*`]`

- Creating a new certificate and adding it to the database with one command:

```
-S -k shortkeyID -y rsa|dsa -n certname -s subject
[-c issuer |-x] -t trustargs [-h tokenname]
[-m serial-number] [-v valid-months] [-w offset-months]
[-d certdir] [-p phone] [-f password-file] [-1] [-2] [-3] [-4]
```

- Making a separate certificate request:

```
-R -k shortkeyID -y rsa|dsa -s subject [-h tokenname]
[-d certdir] [-p phone] [-o output-file] [-f password-file]
```

- Creating a new binary certificate from a binary certificate request:

```
-C [-c issuer |-k shortkeyID -y rsa|dsa -x] [-f password-file]
[-h tokenname] -i cert-request-file -o output-file [-m serial-number]
[-v valid-months] [-w offset-months] [-d certdir] [-1] [-2] [-3]
[-4]
```

- Adding a certificate to an existing database:

```
-A -n certname -t trustargs [-h tokenname] [-d certdir] [-a]
[-i cert-request-file]
```

- Listing all certificates or a named certificate:

```
-L [-n certname] [-d certdir] [-r] [-a]
```

- Validating a certificate:

```
-V -n certname -b validity-time -u certusage [-e] [-l] [-d certdir]
```

- Modifying a certificate's trust attribute:

```
-M -n certname -t trustargs [-d certdir]
```

- Displaying a list of the options and arguments used by Certificate Database Tool:

```
-H
```

Examples

This section contains examples for the following tasks:

- Creating a New Certificate Database
- Listing Certificates in a Database
- Creating a Certificate Request
- Creating a Certificate
- Adding a Certificate to the Database
- Validating a Certificate

Creating a New Certificate Database

This example creates a new certificate database (`cert7.db` file) in the specified directory:

`certutil -N -d` *certdir*

You must generate the associated `key3.db` and `secmod.db` files by using the Key Database Tool or other tools.

Listing Certificates in a Database

This example lists all the certificates in the `cert7.db` file in the specified directory:

`certutil -L -d` *certdir*

Certificate Database Tool displays output similar to the following:

```
Certificate Name              Trust Attributes

Uptime Group Plc. Class 1 CA        C,C,
VeriSign Class 1 Primary CA         ,C,
VeriSign Class 2 Primary CA         C,C,C
AT&T Certificate Services           C,C,
```

```
GTE CyberTrust Secure Server CA     C,,
Verisign/RSA Commercial CA          C,C,
AT&T Directory Services             C,C,
BelSign Secure Server CA            C,,
Verisign/RSA Secure Server CA       C,C,
GTE CyberTrust Root CA              C,C,
Uptime Group Plc. Class 4 CA        ,C,
VeriSign Class 3 Primary CA         C,C,C
Canada Post Corporation CA          C,C,
Integrion CA                        C,C,C
IBM World Registry CA               C,C,C
GTIS/PWGSC, Canada Gov. Web CA      C,C,
GTIS/PWGSC, Canada Gov. Secure CA   C,C,C
MCI Mall CA                         C,C,
VeriSign Class 4 Primary CA         C,C,C
KEYWITNESS, Canada CA               C,C,
BelSign Object Publishing CA        ,,C
BBN Certificate Services CA Root 1  C,C,
p    Valid peer
P    Trusted peer (implies p)
c    Valid CA
T    Trusted CA to issue client certs (implies c)
C    Trusted CA to issue server certs(for ssl only) (implies c)
u    User cert
w    Send warning
```

Creating a Certificate Request

This example generates a binary certificate request file named `e95c.req` in the specified directory:

```
certutil -R -s "CN=John Smith, O=Netscape, L=Mountain View,
ST=California, C=US" -p "650-555-8888" -k e95c -o e95c.req -d
certdir
```

Before it creates the request file, Certificate Database Tool prompts you for a password:

```
Enter Password or Pin for "Communicator Certificate DB":
```

Creating a Certificate

A valid certificate must be issued by a trusted CA. If a CA key pair is not available, you can create a self-signed certificate (for purposes of illustration) with the `-x` argument. This example creates a new, self-signed binary certificate named `e95c.crt`, from a binary certificate request named `e95c.req`, in the specified directory.

```
certutil -C -i e95c.req -o e95c.crt -k e95c -m 1234
-f password-file -x -d certdir
```

The following example creates a new binary certificate named `one.crt`, from a binary certificate request named `one.req`, in the specified directory. It is issued by the self-signed certificate created above, `e95c.crt`.

```
certutil -C -m 2345 -i one.req -o one.crt -c e95c.crt -d certdir
```

Adding a Certificate to the Database

This example adds a certificate to the certificate database:

```
certutil -A -n jsmith@netscape.com -t "C,C,C" -i e95c.crt
-d certdir
```

You can see this certificate in the database with this command:

```
certutil -L -n jsmith@netscape.com -d certdir
```

Certificate Database Tool displays output similar to the following:

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 0 (0x0)
    Signature Algorithm: PKCS #1 MD5 With RSA Encryption
    Issuer: CN=John Smith, O=Netscape, L=Mountain View,
ST=California, C=US
    Validity:
        Not Before: Thu Mar 12 00:10:40 1998
        Not After: Sat Sep 12 00:10:40 1998
Subject: CN=John Smith, O=Netscape, L=Mountain View,
ST=California, C=US
```

```
Subject Public Key Info:
  Public Key Algorithm: PKCS #1 RSA Encryption
  RSA Public Key:
    Modulus:
        00:da:53:23:58:00:91:6a:d1:a2:39:26:2f:06:3a:
        38:eb:d4:c1:54:a3:62:00:b9:f0:7f:d6:00:76:aa:
        18:da:6b:79:71:5b:d9:8a:82:24:07:ed:49:5b:33:
        bf:c5:79:7c:f6:22:a7:18:66:9f:ab:2d:33:03:ec:
        63:eb:9d:0d:02:1b:da:32:ae:6c:d4:40:95:9f:b3:
        44:8b:8e:8e:a3:ae:ad:08:38:4f:2e:53:e9:e1:3f:
        8e:43:7f:51:61:b9:0f:f3:a6:25:1e:0b:93:74:8f:
        c6:13:a3:cd:51:40:84:0e:79:ea:b7:6b:d1:cc:6b:
        78:d0:5d:da:be:2b:57:c2:6f
    Exponent: 65537 (0x10001)
Signature Algorithm: PKCS #1 MD5 With RSA Encryption
Signature:
  44:15:e5:ae:c4:30:2c:cd:60:89:f1:1d:22:ed:5e:5b:10:c8:
  7e:5f:56:8c:b4:00:12:ed:5f:a4:6a:12:c3:0d:01:03:09:f2:
  2f:e7:fd:95:25:47:80:ea:c1:25:5a:33:98:16:52:78:24:80:
  c9:53:11:40:99:f5:bd:b8:e9:35:0e:5d:3e:38:6a:5c:10:d1:
  c6:f9:54:af:28:56:62:f4:2f:b3:9b:50:e1:c3:a2:ba:27:ee:
  07:9f:89:2e:78:5c:6d:46:b6:5e:99:de:e6:9d:eb:d9:ff:b2:
  5f:c6:f6:c6:52:4a:d4:67:be:8d:fc:dd:52:51:8e:a2:d7:15:
  71:3e

Certificate Trust Flags:
  SSL Flags:
    Valid CA
    Trusted CA
  Email Flags:
    Valid CA
    Trusted CA
  Object Signing Flags:
    Valid CA
    Trusted CA
```

Validating a Certificate

This example validates a certificate:

`certutil -V -n jsmith@netscape.com -b 9803201212Z -u SR -e -l -d` *certdir*

Certificate Database Tool shows results similar to

```
Certificate:'jsmith@netscape.com' is valid.
```

or

```
UID=jsmith, E=jsmith@netscape.com, CN=John Smith, O=Netscape
Communications Corp., C=US : Expired certificate
```

or

```
UID=jsmith, E=jsmith@netscape.com, CN=John Smith, O=Netscape
Communications Corp., C=US : Certificate not approved for this
operation
```

Appendix

E

Key Database Tool

Key Database Tool is a command-line utility that can modify the key database file (`key3.db`) of Netscape Certificate Management System (CMS). You can use the utility to create or change the database password, generate new public and private key pairs, display the contents of the database, or delete key pairs from the database.

Key database management tasks are part of a process that typically also involves managing client certificate databases (`cert7.db` file). The key and certificate management process generally begins with creating keys in the key database, then generating and managing certificates in the certificate database.

This appendix discusses key database management. For information on certificate database and security module database management, see "Certificate Database Tool" on page 1197 and the section "modutil" in Appendix B of *Managing Servers with Netscape Console*.

This appendix has the following sections:

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

To run Key Database Tool, type the command

`keyutil` *option* [*arguments*]

where *option* and *arguments* are combinations of the options and arguments listed in the following section. Each command takes one option. Each option may take zero or more arguments. To see a usage string, issue the command without options, or with the `-H` option.

Options and Arguments

Options specify an action and are uppercase. Option arguments modify an action and are lowercase. Key Database Tool options and their arguments are defined as follows:

Options	
`-N`	Create a new key database and set its password. Use the `-h` *tokenname* argument to specify a specific hardware or software token in which to create the new database.
`-C`	Change the password to a key database.
`-G`	Generate a new public and private key pair within a database. The key database should already exist; if one is not present, this option will initialize one by default. Some smart cards (for example, the Litronic card) can store only one key pair. If you create a new key pair for such a card, the previous pair is overwritten.

`-L`	List the keyID of keys in the key database. A keyID is the modulus of the RSA key or the `publicValue` of the DSA key. IDs are displayed in hexadecimal ("0x" is not shown). You can identify keys by a *shortkeyID*. The shortkeyID is the first few bytes of the keyID, starting from the second byte, with a length sufficient to identify it uniquely. Use the `-a` argument to list keys of all tokens. Otherwise the list will contain only keys in the default (internal) slot. Use the `-l` argument to list DSA as well as RSA keys.
`-P`	Display public key information on the screen.
`-D`	Delete a private key from a key database. Specify the key to delete with the `-k` argument. Specify the database from which to delete the key with the `-d` argument. Use the `-t` argument to specify explicitly whether to delete a DSA or an RSA key. If you do not use the `-t` argument, the option looks for an RSA key matching the shortkeyID. When you delete keys, be sure to also remove any certificates associated with those keys from the certificate database, by using the Certificate Database Tool. Some smart cards (for example, the Litronic card) do not let you remove a public key you have generated. In such a case, only the private key is deleted from the key pair. You can display the public key with the command `keyutil -L -h` *tokenname*.
`-H`	Display a list of the options and arguments used by Key Database Tool.
Arguments	
`-a`	List the RSA keys of all tokens when listing keys in the database.

`-d` *keydir*	Specify a directory containing a key database file. On Unix Key Database Tool defaults to `$HOME/.netscape` (that is, `~/.netscape`), and on Windows NT the default is the current directory. The `key3.db` and `cert7.db` database files must reside in the same directory.
`-e` *exp*	Set an alternate exponent value to use in generating a new RSA public key for the database, instead of the default value of `65537`. The available alternate values are `3` and `17`.
`-f` *noise-file*	Read a seed value from the specified binary file to use in generating a new RSA private and public key pair. This argument makes it possible to use hardware-generated seed values and unnecessary to manually create a value from the keyboard. The minimum file size is 20 bytes.
`-h` *tokenname*	Specify the name of a token to act on. Unless otherwise specified, the default token is an internal slot (specifically, internal slot 2). An internal slot is a virtual slot maintained in software, rather than a hardware device. Internal slot 2 is used by key and certificate services. Internal slot 1 is used by cryptographic services. Use the Module Database Tool (`modutil -list`) to get a list of token names in the module database.
`-k` *shortkeyID*	Specify a private key by using the key identifier. You can use the complete keyID (as shown by the `-L` option), or the shortkeyID. The shortkeyID is the first few bytes of the keyID, starting from the second byte, with a length sufficient to identify it uniquely. If you specify a shortkeyID that is not unique, the first private key that matches the shortkeyID is found.
`-l`	List DSA as well as RSA keys when listing keys in the key database.
`-q` *pqgfile*	Read an alternate PQG value from the specified file when generating DSA key pairs. If this argument is not used, Key Database Tool generates its own PQG value. PQG files are created with a separate DSA utility.

`-s` *size*	Set a key size to use when generating new public and private key pairs. The minimum is `256` bits `and` the maximum is `1024 bits.` The default is `1024` bits. Any size between the minimum and maximum is allowed.
`-t rsa|dsa`	Specify the type of a key, either RSA or DSA. The default value is `rsa`. By specifying the type of key you can avoid mistakes caused by duplicate shortkeyIDs.
`-w` *password-file*	Specify a file to automatically supply the password necessary to access a key database. This is a plain-text file containing one password. You should not use this argument if you are accessing an internal slot and hardware tokens that use different passwords. Be sure to prevent unauthorized access to this file.

Usage

Key Database Tool's capabilities are grouped as follows, using these combinations of options and arguments. The specifications in square brackets are optional, those without square brackets are required.

- Creating a new `key3.db` file and setting its password:

  `-N [-d` *keydir*`] [-w` *password-file*`]`

- Changing the password to a key database file:

  `-C [-d` *keydir*`]`

- Generating new RSA key pairs in a key database file:

  `-G [-h` *tokenname*`] [-t rsa] [-s` *num*`] [-e` *exp*`] [-d` *keydir*`]`
  `[-f` *noise-file*`] [-w` *password-file*`]`

- Generating new DSA key pairs in a key database file:

  `-G [-h` *tokenname*`] -t dsa [-q` *pqgfile* `-s` *num*`]`
  `[-d` *keydir*`] [-w` *password-file*`]`

- Listing the keyIDs of the keys in a database:

`-L [-a] [-l] [-t rsa|dsa] [-h` *tokenname*`] [-d` *keydir*`]`

- Displaying public key information from the database:

  `-P -k` *shortkeyID* `[-t rsa|dsa] [-h` *tokenname*`]`
  `[-d` *keydir*`] [-w` *password-file*`]`

- Deleting private keys from a key database file:

  `-D -k` *shortkeyID* `[-t rsa|dsa] [-h` *tokenname*`]`
  `[-d` *keydir*`] [-w` *password-file*`]`

- Displaying a list of the options and arguments used by Key Database Tool:

  `-H`

Examples

Includes the following:

- Creating a Key Database
- Generating a New Key
- Displaying Public Key Information
- Listing Key IDs
- Deleting a Private Key

Creating a Key Database

This example creates new key database files (`key3.db and secmod.db`) in the specified directory:

`keyutil -N -d` *keydir*

Key Database Tool prompts you as follows:

```
Creating a brand new key database:keydir/key3.db
Database not initialized. Setting password.
Enter new password:
```

```
Re-enter password:
```

After you enter the password, Key Database Tool creates new `key3.db` and `secmod.db` files in the specified directory.

Generating a New Key

This example generates a new key in a key database:

`keyutil -G -d` *keydir*

Key Database Tool then displays the following:

```
------------------------------------------------
Netscape Communications Corporation
Key Generation
------------------------------------------------

Welcome to the key generator. With this program, you can
generate the public and private keys that you use for secure
communications.

A random seed must be generated that will be used in the
creation of your key. One of the easiest ways to create a random
seed is to use the timing of keystrokes on a keyboard.

You have specified the name 'mykey' for your key

If this is correct, press enter:

To begin, type keys on the keyboard until this progress meter is
full. DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

|************************************************************|

Finished. Press enter to continue:

Generating key. This may take a few moments...

Password:

generated public/private key pair
```

Note that if you do not specify a token name, the key is generated on the internal slot. This is equivalent to the `-h internal` argument.

If you use the `-f` *noise-file* argument, Key Database Tool does not ask for keyboard input.

If you use the `-w` *password-file* argument, Key Database Tool reads the password from the file instead of asking for keyboard input. Avoid using this argument when you are accessing both the internal slot and tokens that have different passwords.

Displaying Public Key Information

This example prints the public key's information:

`keyutil -P -k e95c -d` *keydir*

The public key information appears after you give the correct password:

```
Password:

It's the first key found.
RSA Public-Key:
modulus:
00:e9:5c:4a:73:74:39:22:6d:c6:da:4e:b3:1f:01:26:9d:be:
d1:74:ae:cd:c7:7d:65:f9:1d:31:1f:71:fb:60:d0:45:46:5f:
5a:19:e7:61:1e:e7:ce:9f:4a:13:4e:d6:e9:06:90:2a:ba:bd:
0b:5f:7b:a3:28:21:1e:0f:1c:f4:3a:ba:3a:8f:0b:e1:99:91:
cc:e8:fd:17:d2:1c:66:13:6b:95:27:b1:eb:bc:9c:e6:7b:f0:
3a:b9:44:dc:24:a6:f8:83:9a:9e:80:3f:74:48:09:6b:3f:a6:
46:51:be:e0:1b:51:87:8c:44:94:f0:fe:41:fe:b4:9f:4c:0a:
04:a9:a1
publicExponent: 65537 (0x10001)
```

Listing Key IDs

This command lists the key IDs in the key database:

`keyutil -L -d` *keydir*

After you enter the password, Key Database Tool displays the following:

```
RSA Public-Key:
modulus:
00:e9:5c:4a:73:74:39:22:6d:c6:da:4e:b3:1f:01:26:9d:be:
d1:74:ae:cd:c7:7d:65:f9:1d:31:1f:71:fb:60:d0:45:46:5f:
```

```
5a:19:e7:61:1e:e7:ce:9f:4a:13:4e:d6:e9:06:90:2a:ba:bd:
0b:5f:7b:a3:28:21:1e:0f:1c:f4:3a:ba:3a:8f:0b:e1:99:91:
cc:e8:fd:17:d2:1c:66:13:6b:95:27:b1:eb:bc:9c:e6:7b:f0:
3a:b9:44:dc:24:a6:f8:83:9a:9e:80:3f:74:48:09:6b:3f:a6:
46:51:be:e0:1b:51:87:8c:44:94:f0:fe:41:fe:b4:9f:4c:0a:
04:a9:a1
```

When unmodified, this command lists all the RSA keys in the default (internal) slot. You can refine this command's output with the `-a`, `-h`, and `-l` arguments.

Deleting a Private Key

This example deletes a private key from the key database:

`keyutil -D -k e95c -d` *keydir*

When you delete keys, be sure to remove any certificates associated with those keys from the certificate database by using the Certificate Database Tool.

Appendix

F

Netscape Signing Tool

This appendix describes how to use version 1.3 of Netscape Signing Tool (`signtool` on the command line) to digitally sign software, including binary files intended for distribution via SmartUpdate, Java class files, and JavaScript scripts. Version 1.3 includes all the capabilities of, and is fully compatible with, previous versions of Netscape Signing Tool (.50, .60, 1.0, 1.1, and 1.2).

This appendix has these sections:

Introduction to Netscape Signing Tool

This section reviews basic concepts that you need to understand before you begin using version 1.3 of Netscape Signing Tool to sign files or JavaScript scripts. If you are already familiar with object-signing concepts, go straight to "Using Netscape Signing Tool" on page 1226.

- What Is Netscape Signing Tool?
- JAR Format and JAR Archives
- What Signing a File Means
- Object-Signing Certificates

For a complete introduction to object signing technology, see *Netscape Object Signing: Establishing Trust for Downloaded Software* at this URL:

```
http://developer.netscape.com/docs/manuals/signedobj/trust/
index.htm
```

What Is Netscape Signing Tool?

Netscape Signing Tool is a stand-alone command-line tool that creates digital signatures and uses the Java Archive (JAR) format to associate them with files in a directory. It is intended for use by system administrators and by developers who want to distribute software electronically over the Internet.

Netscape Signing Tool 1.3 is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

This appendix describes how to use Netscape Signing Tool 1.3 to sign Java applets, JavaScript scripts, plug-ins, and other files and how to package the signed objects in a *JAR archive* (also called a *JAR file*), which is a digital envelope for a compressed collection of files. Communicator client software uses JAR archives to install or update software automatically.

Electronic software distribution over any network involves potential security problems. To help address some of these problems, you can associate digital signatures with the files in a JAR archive. Digital signatures allow Communicator to perform two operations that are important to end users:

- Confirm the identity of the individual, company, or other entity whose digital signature is associated with the files
- Check whether the files have been tampered with since being signed

You do not need to understand the technical details of JAR archives or digital signatures to use Netscape Signing Tool. However, you do need some familiarity with the concepts described in the rest of this appendix. If you are already familiar with basic object-signing concepts, go straight to "Using Netscape Signing Tool" on page 1226.

JAR Format and JAR Archives

The *Java Archive (JAR) format* **is** a set of conventions for associating digital signatures, installer scripts, and other information with files in a directory. Signing tools such as Netscape Signing Tool allow you to sign files using the JAR format and package them as a single JAR file. JAR files are used by Communicator client software to support automatic software installation, user-controlled access to local system resources by Java applets, and other features that help address potential security problems.

The *JAR file type* is a registered Internet MIME type based on the standard cross-platform ZIP archive format. A JAR file functions as a digital envelope for a compressed collection of files. The JAR file type is distinct from the JAR format, which is simply a way of organizing information in a directory.

Because the JAR format doesn't require a digital signature to be stored physically inside the file with which it is associated, JAR files are extremely flexible. You can use Netscape Signing Tool to sign any files, including Java class files, Communicator plug-ins, or any other kind of document or application. You can also use version 1.1 and later versions of Netscape Signing Tool to sign inline JavaScript scripts.

You must create a JAR file if you want to take advantage of Communicator's SmartUpdate feature. Communicator can automatically locate, download, and install components, plug-ins, and Java classes on a user's machine, thus freeing the user from this chore. Automatic software installation also helps both

software developers who want to distribute software and updates over the Internet and system administrators using Mission Control to manage a corporate intranet.

You don't need to know anything else about the JAR format to use Netscape Signing Tool, which takes care of the details for you. For detailed information about the JAR format, see *The Jar Format* at this URL:

```
http://developer.netscape.com/docs/manuals/signedobj/jarfile/
index.html
```

For detailed information about using the JAR Installation Manager to package your software for use with SmartUpdate, see *Using JAR Installation Manager for SmartUpdate* at this URL:

```
http://developer.netscape.com/docs/manuals/communicator/jarman/
index.htm
```

What Signing a File Means

If you have a signing certificate, you can use Netscape Signing Tool to digitally sign files and package them as a JAR file**.** An *object-signing certificate* is a special kind of certificate that allows you to associate your digital signature with one or more files. For information about obtaining an object-signing certificate, see *Object-Signing Tools* at this URL:

```
http://developer.netscape.com/docs/manuals/
index.html?content=security.html
```

An individual file can potentially be signed with multiple digital signatures. For example, a commercial software developer might sign the files that constitute a software product to prove that the files are indeed from a particular company. A network administrator manager might sign the same files with an additional digital signature based on a company-generated certificate to indicate that the product is approved for use within the company.

The significance of a digital signature is comparable to the significance of a handwritten signature. Once you have signed a file, it is difficult to claim later that you didn't sign it. In some situations, a digital signature may be considered as legally binding as a handwritten signature. Therefore, you should take great care to ensure that you can stand behind any file you sign and distribute.

For example, if you are a software developer, you should test your code to make sure it is virus-free before signing it. Similarly, if you are a network administrator, you should make sure, before signing any code, that it comes from a reliable source and will run correctly with the software installed on the machines to which you are distributing it.

Object-Signing Certificates

Before you can use Netscape Signing Tool to sign files, you must have an object-signing certificate, which is a special certificate whose associated private key is used to create digital signatures.

For testing purposes only, you can create an object-signing certificate with Netscape Signing Tool 1.3. When testing is finished and you are ready to disitribute your software, you should obtain an object-signing certificate from one of two kinds of sources:

- An independent certificate authority (CA) that authenticates your identity and charges you a fee. You typically get a certificate from an independent CA if you want to sign software that will be distributed over the Internet.
- CA server software running on your corporate intranet or extranet. Netscape Certificate Management System provides a complete management solution for creating, deploying, and managing certificates, including CAs that issue object-signing certificates.

You must also have a certificate for the CA that issues your signing certificate before you can sign files. If the certificate authority's certificate isn't already installed in your copy of Communicator, you typically install it by clicking the appropriate link on the certificate authority's web site, for example on the page from which you initiated enrollment for your signing certificate. This is the case for some test certificates, as well as certificates issued by Netscape Certificate Management System: you must download the the CA certificate in addition to obtaining your own signing certificate. CA certificates for several certificate authorities are preinstalled in the Communicator certificate database.

When you receive an object-signing certificate for your own use, it is automatically installed in your copy of the Communicator client software. Communicator supports the public-key cryptography standard known as PKCS #12, which governs key portability. You can, for example, move an object-

signing certificate and its associated private key from one computer to another on a credit-card-sized device called a smart card. For motr information, see "Using Netscape Signing Tool with Smart Cards" on page 1243.

Using Netscape Signing Tool

This section describes how to use Netscape Signing Tool to create digital signatures for files in a directory and to associate the signatures with the files according to the JAR format. Netscape Signing Tool also provides an option that automatically creates a JAR file containing the directory; this option was not implemented in pre-1.0 versions. For maximum flexibility, and for compatibility with scripts that used earlier versions of Netscape Signing Tool, you can still use a ZIP utility to create the JAR file.

- Getting Ready to Use Netscape Signing Tool
- Signing a File
- Using Netscape Signing Tool with a ZIP Utility
- Tips and Techniques

For a complete list of Netscape Signing Tool command-line options, see "SignTool Syntax and Options" on page 1232.

Getting Ready to Use Netscape Signing Tool

Before using Netscape Signing Tool, you must have the `signtool` executable in your path environment variable. You must also have an object-signing certificate.

Netscape Signing Tool includes an option that allows you to generate an object-signing certificate for testing purposes. For information about using this option, see "Generating Test Object-Signing Certificates" on page 1241.

Although suitable for testing purposes, the object-signing certificate produced by Netscape Signing Tool is not recommended for signing finished software that will be widely distributed over the Internet or an intranet. When you are

ready to sign finished software, you will need to get an object-signing certificate from your company's internal certificate authority, if it has one, or from a third-party certificate authority.

The sections that follow describe how to prepare Netscape Signing Tool for signing files.

Setting Up Your Certificate

These instructions apply to an object-signing certificate obtained from a third party or an in-house CA for use in signing finished code. During development, you may wish to use a special certificate generated by Netscape Signing Tool for testing purposes.

If you obtained your object-signing certificate while running Communicator on a system that's different from the system on which you intend to sign files, you need to copy your certificate and private key files to the new system. Communicator's certificate and key databases are portable among all platforms.

On the computer where you ran Communicator to get the object-signing certificate, locate the files `key3.db` and `cert7.db`. For example, on a typical Windows NT system, these files are found at `C:\Program Files\NETSCAPE\USERS\username\`. You must copy these files to the system where you intend to sign pages. (If you use FTP, be sure to transfer in binary mode.)

If you are running Netscape Signing Tool on a Unix system and you don't already have a `~/.netscape` directory, first run Communicator once to create one. If you want to maintain whatever certificates are already in your `~/.netscape` directory, put the existing `key3.db` and `cert7.db` files in some other directory before replacing them with the versions that include the object-signing certificate you want to use with Netscape Signing Tool.

If you are using Unix, set up an alias to call `signtool`, or place it in your path.

If you are using Windows 95 or NT, the `signtool` executable doesn't know where your certificates are, so either put the `key3.db` and `cert7.db` files in the current directory and use "`-d.`" or use `-d` to point to the directory in which they are located.

Warning Keep copies of the `key3.db` and `cert7.db` files somewhere separate from the copies you use with the `signtool` executable. This ensures that you won't lose your certificates if you accidentally damage the files. If your keys are on external tokens, such as smart cards, you should keep a copy the `secmod.db` file.

Listing Available Certificates

You use the `-L` option to list the nicknames for all available certificates and check which ones are signing certificates, as shown in this Unix example:

```
% signtool -L
using certificate directory: /u/jsmith/.netscape
S Certificates
- ------------
  BBN Certificate Services CA Root 1
  IBM World Registry CA
  VeriSign Class 1 CA - Individual Subscriber - VeriSign, Inc.
  GTE CyberTrust Root CA
  Uptime Group Plc. Class 4 CA
* Verisign Object Signing Cert
  Integrion CA
  GTE CyberTrust Secure Server CA
  AT&T Directory Services
* test object signing cert
  Uptime Group Plc. Class 1 CA
  VeriSign Class 1 Primary CA
- ------------
Certificates that can be used to sign objects have *'s to their left.
%
```

In the above example, two signing certificates are displayed: `Verisign Object Signing Cert` and `test object signing cert`.

You use the `-l` option to get a list of signing certificates only, including the signing CA for each, as shown in this Unix example:

```
% signtool -l
using certificate directory: /u/jsmith/.netscape
Object signing certificates
-------------------------------------
Verisign Object Signing Cert
    Issued by: VeriSign, Inc. - Verisign, Inc.
    Expires: Tue May 19, 1998
test object signing cert
    Issued by: test object signing cert (Signtool 1.0 Testing
Certificate (960187691))
    Expires: Sun May 17, 1998
```

```
-------------------------------------
For a list including CAs, use "signtool -L"
```

Signing a File

To sign a file using Netscape Signing Tool, follow these steps:

1. Create an empty directory.

```
% mkdir signdir
```

2. Put some file into it.

```
% echo boo > signdir/test.f
```

3. Specify the name of your object-signing certificate and sign the directory.

If you are using Unix, this example assumes you have put your `.db` files in the `~/.netscape` directory, as explained in "Setting Up Your Certificate" on page 1227.

```
% signtool -k MySignCert -Z testjar.jar signdir

using key "MySignCert"
using certificate directory: /u/jsmith/.netscape
Generating signdir/META-INF/manifest.mf file..
--> test.f
adding signdir/test.f to testjar.jar
Generating signtool.sf file..
Enter Password or Pin for "Communicator Certificate DB":
```

4. At the prompt, type the password to your private-key database.

If it accepts the password, `signtool` responds as follows:

```
adding signdir/META-INF/manifest.mf to testjar.jar
adding signdir/META-INF/signtool.sf to testjar.jar
adding signdir/META-INF/signtool.rsa to testjar.jar
tree "signdir" signed successfully
```

5. Test the archive you just created.

```
% signtool -v testjar.jar

using certificate directory: /u/jsmith/.netscape
archive "testjar.jar" has passed crypto verification.
```

```
          status   path
------------   -------------------
    verified   test.f
```

You can also use Netscape Signing Tool from within a script to automate some aspects of signing. For example, here's a Windows script that starts with an unsigned JAR file, unpackages it, signs it, and then repackages it:

```
rem Expand the jar file into a new directory
unzip -qq myjar.jar -d signjar
del myjar.jar
rem Sign everything in the new directory and recompress
signtool -k MySignCert -Z myjar.jar signdir
```

Using Netscape Signing Tool with a ZIP Utility

To use Netscape Signing Tool with a ZIP utility, you must have the utility in your path environment variable. You should use the `zip.exe` utility rather than `pkzip.exe`, which cannot handle long filenames.

You can use a ZIP utility instead of the `-Z` option to package a signed archive into a JAR file after you have signed it:

```
% cd signdir
% zip -r ../myjar.jar *
  adding: META-INF/ (stored 0%)
  adding: META-INF/manifest.mf (deflated 15%)
  adding: META-INF/signtool.sf (deflated 28%)
  adding: META-INF/signtool.rsa (stored 0%)
  adding: text.txt (stored 0%)
%
```

Tips and Techniques

- If you are storing JAR files or their components in CVS, store them as binary files.
- If you are signing metadata only and not files, you still need to create a blank directory for Netscape Signing Tool to sign.

- When using the Windows NT version of Netscape Signing Tool, always use a relative path.
- The command `signtool -L` should list your object-signing certificate with an asterisk (`*`) beside it. If it doesn't, you cannot sign files.
- If you are having problems using the JAR file from within Navigator, check `signtool -v` on your final archive.
- Don't use Netscape Signing Tool's `-G` option while Communicator is running. The `-G` option writes to the security databases as it generates certificates, and corruption could occur if Communicator simultaneously attempts to write to these files. All other Netscape Signing Tool options are read-only and can't harm these files.
- If you see the error `Unknown issuer`, you need to get the certificate of the certificate authority that issued your signing certificate. Alternatively, you may have the certificate authority's certificate, but it may not be trusted for object signing.
- If you see the error `Issuer not trusted`, open Communicator on the system you used when you obtained the certificate and follow these steps:

  1. Click the Security button in a Navigator window.
  2. Click Signers under Certificates in the left frame.
  3. Select the CA that issued your signing certificate.
  4. Click the Edit button.
  5. Select the "Accept this Certificate Authority for Certifying software developers" checkbox.
  6. Click OK.

  You then need to transfer the `cert7.db` file to the appropriate directory on the system on which you are running Netscape Signing Tool, as described in "Setting Up Your Certificate" on page 1227.

SignTool Syntax and Options

This section summarizes the syntax and options for Netscape Signing Tool 1.3.

- Command Syntax
- Command Options
- Command File Syntax
- Command File Keywords and Example

Command Syntax

To run Netscape Signing Tool, type

`signtool` *options*

where *options* can be any sequence of the options listed in this chapter.

Each argument for each `signtool` option must be in quotes if it contains any spaces or other nonalphanumeric characters.

Command Options

Options for `signtool` are defined as follows:

Options	
`-b` *basename*	Specifies the base filename for the `.rsa` and `.sf` files in the `META-INF` directory (required by The JAR Format). For example, `-b signatures` causes the files to be named `signatures.rsa` and `signatures.sf`. The default is `signtool`. The `-b` option is available in Netscape Signing Tool 1.0 and later versions only.

`-c#`	Specifies the compression level for the `-J` or `-Z` option. The symbol # represents a number from 0 to 9, where 0 means no compression and 9 means maximum compression. The higher the level of compression, the smaller the output but the longer the operation takes. If the `-c#` option is not used with either the `-J` or the `-Z` option, the default compression value used by both the `-J` and `-Z` options is 6.
`-d` *certdir*	Specifies your certificate database directory; that is, the directory in which you placed your `key3.db` and `cert7.db` files. To specify the current directory, use "`-d.`" (including the period). The Unix version of `signtool` assumes `~/.netscape` unless told otherwise. The NT version of `signtool` always requires the use of the -d option to specify where the database files are located.
`-e` *extension*	Tells `signtool` to sign only files with the given extension; for example, use `-e".class"` to sign only Java class files. Note that with Netscape Signing Tool version 1.1 and later this option can appear multiple times on one command line, making it possible to specify multiple file types or classes to include.
`-f` *commandfile*	Specifies a text file containing Netscape Signing Tool options and arguments in *keyword*=*value* format. All options and arguments can be expressed through this file. For more information about the syntax used with this file, see "Tips and Techniques" on page 1230.
`-i` *scriptname*	Specifies the name of an installer script for SmartUpdate. This script installs files from the JAR archive in the local system after SmartUpdate has validated the digital signature. For more details, see the description of `-m` that follows. The `-i` option provides a straightforward way to provide this information if you don't need to specify any metadata other than an installer script.

`-j` *directory*	Specifies a special JavaScript directory. This option causes the specified directory to be signed and tags its entries as inline JavaScript. This special type of entry does not have to appear in the JAR file itself. Instead, it is located in the HTML page containing the inline scripts. When you use `signtool -v`, these entries are displayed with the string `NOT PRESENT`.
`-k` *key* `...` *directory*	Specifies the nickname (*key*) of the certificate you want to sign with and signs the files in the specified directory. The directory to sign is always specified as the last command-line argument. Thus, it is possible to write `signtool -k MyCert -d . signdir` You may have trouble if the nickname contains a single quotation mark. To avoid problems, escape the quotation mark using the escape conventions for your platform. It's also possible to use the `-k` option without signing any files or specifying a directory. For example, you can use it with the `-l` option to get detailed information about a particular signing certificate.

`-G` *nickname*	Generates a new private-public key pair and corresponding object-signing certificate with the given nickname. The newly generated keys and certificate are installed into the key and certificate databases in the directory specified by the `-d` option. With the NT version of Netscape Signing Tool, you must use the `-d` option with the `-G` option. With the Unix version of Netscape Signing Tool, omitting the `-d` option causes the tool to install the keys and certificate in the Communicator key and certificate databases. If you are installing the keys and certificate in the Communicator databases, you must exit Communicator before using this option; otherwise, you risk corrupting the databases. In all cases, the certificate is also output to a file named `x509.cacert`, which has the MIME-type `application/x-x509-ca-cert`. Unlike certificates normally used to sign finished code to be distributed over a network, a test certificate created with `-G` is not signed by a recognized certificate authority. Instead, it is self-signed. In addition, a single test signing certificate functions as both an object-signing certificate and a CA. When you are using it to sign objects, it behaves like an object-signing certificate. When it is imported into browser software such as Communicator, it behaves like an object-signing CA and cannot be used to sign objects. The `-G` option is available in Netscape Signing Tool 1.0 and later versions only. By default, it produces only RSA certificates with 1024-byte keys in the internal token. However, you can use the `-s` option specify the required key size and the `-t` option to specify the token. For more information about the use of the `-G` option, see “Generating Test Object-Signing Certificates”“Generating Test Object-Signing Certificates” on page 1241.

`-l`	Lists signing certificates, including issuing CAs. If any of your certificates are expired or invalid, the list will so specify. This option can be used with the `-k` option to list detailed information about a particular signing certificate. The `-l` option is available in Netscape Signing Tool 1.0 and later versions only.
`-J`	Signs a directory of HTML files containing JavaScript and creates as many archive files as are specified in the HTML tags. Even if `signtool` creates more than one archive file, you need to supply the key database password only once. The `-J` option is available only in Netscape Signing Tool 1.0 and later versions. The `-J` option cannot be used at the same time as the `-Z` option. If the `-c#` option is not used with the `-J` option, the default compression value is 6. Note that versions 1.1 and later of Netscape Signing Tool correctly recognizes the `CODEBASE` attribute, allows paths to be expressed for the `CLASS` and `SRC` attributes instead of filenames only, processes LINK tags and parses HTML correctly, and offers clearer error messages.
`-L`	Lists the certificates in your database. An asterisk appears to the left of the nickname for any certificate that can be used to sign objects with `signtool`.
`--leavearc`	Retains the temporary `.arc` (archive) directories that the `-J` option creates. These directories are automatically erased by default. Retaining the temporary directories can be an aid to debugging.

`-m` *metafile*	Specifies the name of a metadata control file. Metadata is signed information attached either to the JAR archive itself or to files within the archive. This metadata can be any ASCII string, but is used mainly for specifying an installer script.
	
	The metadata file contains one entry per line, each with three fields:
	
	field #1: file specification, or `+` if you want to specify global metadata (that is, metadata about the JAR archive itself or all entries in the archive)
	field #2: the name of the data you are specifying; for example: `Install-Script`
	field #3: data corresponding to the name in field #2
	
	For example, the `-i` option uses the equivalent of this line:
	
	`+ Install-Script: script.js`
	
	This example associates a MIME type with a file:
	
	`movie.qt MIME-Type: video/quicktime`
	For information about the way installer script information appears in the manifest file for a JAR archive, see The JAR Format on Netscape DevEdge.

`-M`	Lists the PKCS #11 modules available to `signtool`, including smart cards. The `-M` option is available in Netscape Signing Tool 1.0 and later versions only. For information on using Netscape Signing Tool with smart cards, see "Using Netscape Signing Tool with Smart Cards" on page 1243. For information on using the `-M` option to verify FIPS-140-1 validated mode, see "Netscape Signing Tool and FIPS-140-1" on page 1247.
`--norecurse`	Blocks recursion into subdirectories when signing a directory's contents or when parsing HTML.

`-o`	Optimizes the archive for size. Use this *only* if you are signing very large archives containing hundreds of files. This option makes the manifest files (required by the JAR format) considerably smaller, but they contain slightly less information.
`--outfile` *outputfile*	Specifies a file to receive redirected output from Netscape Signing Tool.
`-p` *password*	Specifies a password for the private-key database. Note that the password entered on the command line is displayed as plain text.
`-s` *keysize*	Specifies the size of the key for generated certificate. Use the `-M` option to find out what tokens are available. The `-s` option can be used with the `-G` option only.
`-t` *token*	Specifies which available token should generate the key and receive the certificate. Use the `-M` option to find out what tokens are available. The `-t` option can be used with the `-G` option only.
`-v` *archive*	Displays the contents of an archive and verifies the cryptographic integrity of the digital signatures it contains and the files with which they are associated. This includes checking that the certificate for the issuer of the object-signing certificate is listed in the certificate database, that the CA's digital signature on the object-signing certificate is valid, that the relevant certificates have not expired, and so on.
`--verbosity` *value*	Sets the quantity of information Netscape Signing Tool generates in operation. A value of `0` (zero) is the default and gives full information. A value of `-1` suppresses most messages, but not error messages.
`-w` *archive*	Displays the names of signers of any files in the archive.
`-x` *directory*	Excludes the specified directory from signing. Note that with Netscape Signing Tool version 1.1 and later this option can appear multiple times on one command line, making it possible to specify several particular directories to exclude.

`-z`	Tells `signtool` not to store the signing time in the digital signature. This option is useful if you want the expiration date of the signature checked against the current date and time rather than the time the files were signed.
`-Z` *jarfile*	Creates a JAR file with the specified name. You must specify this option if you want `signtool` to create the JAR file; it does not do so automatically. If you don't specify `-Z`, you must use an external ZIP tool to create the JAR file. The `-Z` option cannot be used at the same time as the `-J` option. If the `-c#` option is not used with the `-Z` option, the default compression value is 6.

Command File Syntax

Entries in a Netscape Signing Tool command file have this general format:

keyword=value

Everything before the = sign on a single line is a keyword, and everything from the = sign to the end of line is a value. The value may include = signs; only the first = sign on a line is interpreted. Blank lines are ignored, but white space on a line with keywords and values is assumed to be part of the keyword (if it comes before the equal sign) or part of the value (if it comes after the first equal sign). Keywords are case insensitive, values are generally case sensitive. Since the = sign and newline delimit the value, it should not be quoted.

Command File Keywords and Example

Here are the command file keywords and their values:

Keyword	Value
`basename`	Same as `-b` option.
`compression`	Same as `-c` option.
`certdir`	Same as `-d` option.
`extension`	Same as `-e` option.
`generate`	Same as `-G` option.
`installscript`	Same as `-i` option.
`javascriptdir`	Same as `-j` option.
`htmldir`	Same as `-J` option.
`certname`	Nickname of certificate, as with `-k` and `-l -k` options.
`signdir`	The directory to be signed, as with `-k` option.
`list`	Same as `-l` option. Value is ignored, but = sign must be present.
`listall`	Same as `-L` option. Value is ignored, but = sign must be present.
`metafile`	Same as `-m` option.
`modules`	Same as `-M` option. Value is ignored, but = sign must be present.
`optimize`	Same as `-o` option. Value is ignored, but = sign must be present.
`password`	Same as `-p` option.
`keysize`	Same as `-s` option.
`token`	Same as `-t` option.
`verify`	Same as `-v` option.
`who`	Same as `-w` option.
`exclude`	Same as `-x` option.
`notime`	Same as `-z` option. value is ignored, but = sign must be present.

Keyword	Value
`jarfile`	Same as `-Z` option.
`outfile`	Name of a file to which output and error messages will be redirected. This option has no command-line equivalent.

Here's an example of the use of the command file. The command

```
signtool -d c:\netscape\users\james -k mycert -Z myjar.jar signdir > output.txt
```

becomes

`signtool -f` *somefile*

where *somefile* contains the following lines:

```
certdir=c:\netscape\users\james
certname=mycert
jarfile=myjar.jar
signdir=signdir
outfile=output.txt
```

Generating Test Object-Signing Certificates

Netscape Signing Tool allows you to create object-signing certificates for testing purposes. This section describes how to create and use such test certificates.

Unlike certificates normally used to sign finished code to be distributed over a network, the test certificates created with Netscape Signing Tool are not signed by a recognized certificate authority. Instead, they are self-signed. In addition, a single test signing certificate functions as both an object-signing certificate and a CA. When you are using it to sign objects, it behaves like an object-signing certificate. When it is imported into browser software such as Communicator, it behaves like an object-signing CA.

Generating the Keys and Certificate

The `signtool` option `-G` generates a new public-private key pair and certificate. It takes the nickname of the new certificate as an argument. The newly generated keys and certificate are installed into the key and certificate databases in the directory specified by the `-d` option. With the NT version of Netscape Signing Tool, you must use the `-d` option with the `-G` option. With the Unix version of Netscape Signing Tool, omitting the `-d` option causes the tool to install the keys and certificate in the Communicator key and certificate databases. In all cases, the certificate is also output to a file named `x509.cacert`, which has the MIME-type `application/x-x509-ca-cert`.

Important Before installing new keys and certificates in the key and certificate databases, you must set the database password (if you have not done so already). To set the password for the key and certificate databases currently being used by Communicator, click the Security icon in the Communicator toolbar, click Passwords, and click Set Password to create a password.

Warning If you intend to install the new key pair and certificate in the Communicator databases, you must exit Communicator before using Netscape Signing Tool to generate the object-signing certificate. Otherwise, you risk corrupting your certificate and key databases.

Certificates contain standard information about the entity they identify, such as the common name and organization name. Netscape Signing Tool prompts you for this information when you run the command with the `-G` option. However, all of the requested fields are optional for test certificates. If you do not enter a common name, the tool provides a default name. In the following example, the user input is in boldface:

```
% signtool -G MyTestCert
using certificate directory: /u/someuser/.netscape
Enter certificate information. All fields are optional. Acceptable
characters are numbers, letters, spaces, and apostrophes.
certificate common name: Test Object Signing Certificate
organization: Netscape Communications Corp.
organization unit: Server Products Division
state or province: California
country (must be exactly 2 characters): US
username: someuser
email address: someuser@netscape.com
Enter Password or Pin for "Communicator Certificate DB": [Password will
not echo]
generated public/private key pair
certificate request generated
```

```
certificate has been signed
certificate "MyTestCert" added to database
Exported certificate to x509.raw and x509.cacert.
%
```

The certificate information is read from standard input. Therefore, the information can be read from a file using the redirection operator (<) in some operating systems. To create a file for this purpose, enter each of the seven input fields, in order, on a separate line. Make sure there is a newline character at the end of the last line. Then run `signtool` with standard input redirected from your file as follows:

```
% signtool -G MyTestCert <inputfile
```

The prompts show up on the screen, but the responses will be automatically read from the file. The password will still be read from the console unless you use the `-p` option to give the password on the command line.

Using Netscape Signing Tool with Smart Cards

This section describes how to use smart cards from within Netscape Signing Tool to digitally sign files.

- What Is a Smart Card?
- Setting Up a Smart Card
- Using the -M Option to List Smart Cards
- Using Netscape Signing Tool and a Smart Card to Sign Files

What Is a Smart Card?

A *smart card* (sometimes called a *token*) is a credit-card-sized card, a key, or other easily removable device that can be used for cryptographic operations and for storing certificates. Smart cards are portable and must be physically inserted in an appropriate smart card reader attached to a computer for use with Communicator software running on that computer. Smart cards extend the

private-key protection provided by Communicator, since private keys stored on the card require the card's presence as well as the password to the private-key database.

Navigator and Netscape Signing Tool support PKCS #11, a cryptographic standard developed to support services provided by smart cards. Before purchasing a smart card for use with Communicator, you should ensure that your vendor provides a PKCS #11 driver that has been tested with Communicator on your platform. Tested brands include Litronic Netsign and Datakey's SignaSURE.

Setting Up a Smart Card

Connect the smart card reader according to the manufacturer instructions. You may need to reset the smart card to a default state using the manufacturer's configuration utility. Not all smart cards require this step.

Smart cards designed for use with Communicator come with a software driver that you should install in your computer according to the manufacturer's instructions. You can then add the driver (also called a *cryptographic module*) to Communicator as follows:

1. Make sure the smart card is inserted in the smart card reader.
2. Click the Security button near the top of a Navigator window.
3. Click Cryptographic Modules in the left frame.
4. Click the Add button.
5. Type an appropriate name for the module you want to add in the box labeled Security Module Name.
6. Type the name of the driver that was supplied with your smart card in the box labeled Security Module File. For Windows systems, this is a dynamic linked library (DLL). You don't have to type the entire path, but you may.
7. Click OK.
8. If Communicator asks for it, type the smart card password.
9. Select the module you've just installed and click the View/Edit button.

10. Make sure the displayed information is correct for the smart card you just installed.

11. Select the name of the smart card.

12. Click the More Info button and examine that information as well.

13. If the state of the smart card (shown near the bottom of the More Info window) is Not Logged In, click OK and then click the Login button. Otherwise, just click OK. (Logging in allows you to install your signing certificate on the smart card. The smart card doesn't have to be logged in within Communicator for you to use it with Netscape Signing Tool.)

14. Click OK again.

After you have activated the smart card, use Communicator to visit the web site for the certificate authority (CA) you want to use and request a signing certificate.

When you submit your information to the certificate authority, Communicator asks you to select the card or database you wish to use to generate your private key. You should select the name of your smart card.

Your system then generates a public-private key pair and submits your request to the CA. When you receive the certificate, it is installed directly onto the card and travels with that smart card. However, you will be unable to use the certificate unless the smart card is inserted in the appropriate reader and you have entered its password correctly.

Using the -M Option to List Smart Cards

You can use the `-M` option to list the PKCS #11 modules, including smart cards, that are available to `signtool`:

```
% signtool -d "c:\netscape\users\jsmith" -M
using certificate directory: c:\netscape\users\<username>
Listing of PKCS11 modules
-----------------------------------------------
   1. Netscape Internal PKCS #11 Module
           (this module is internally loaded)
           slots: 2 slots attached
           status: loaded
     slot: Communicator Internal Cryptographic Services Version 4.0
```

```
    token: Communicator Generic Crypto Svcs
     slot: Communicator User Private Key and Certificate Services
    token: Communicator Certificate DB
  2. CryptOS
          (this is an external module)
DLL name: core32
   slots: 1 slots attached
  status: loaded
    slot: Litronic 210
   token:
  3. Datakey SignaSURE
          (this is an external module)
DLL name: dkck232.dll
   slots: 1 slots attached
  status: loaded
   slot: Datakey Reader
   token: <username>
  -----------------------------------------------
```

Using Netscape Signing Tool and a Smart Card to Sign Files

Before you try to use Netscape Signing Tool with a smart card, try using it to sign a file without a smart card as described in "Using Netscape Signing Tool" on page 1226.

The `signtool` command normally takes an argument of the `-k` option to specify a signing certificate. To sign with a smart card, you supply only the fully qualified name of the certificate.

To see fully qualified certificate names when you run Communicator, click the Security button in Navigator, then click Yours under Certificates in the left frame. Fully qualified names are of the format *smart card:certificate*, for example "`MyCard:My Signing Cert`". You use this name with the `-k` argument as follows:

`signtool -k "MyCard:My Signing Cert"` *directory*

where *directory* is the directory tree you want to sign. `signtool` asks you for two passwords: the password that protects the Communicator certificate database and the password that protects your smart card. If the passwords are correct, `signtool` signs the files in the directory.

Netscape Signing Tool and FIPS-140-1

This section describes how to use Netscape Signing Tool in FIPS-140-1 validated mode. FIPS 140-1 is a U.S. government standard for implementations of cryptographic modules--that is, hardware or software that encrypts and decrypts data or performs other cryptographic operations (such as creating or verifying digital signatures). Many products sold to the U.S. government must comply with one or more of the FIPS standards.

- Using FIPS-140 Mode
- Verifying FIPS Mode

For general information on FIPS standards and Netscape FIPS-140-1 validation, see the FIPS 140-1 FAQ.

Using FIPS-140 Mode

Netscape Signing Tool is FIPS-140-1 validated when it uses the FIPS-validated Netscape cryptographic module. The FIPS module can be activated and deactivated from within Communicator. Communicator stores the module choice in the security module database (called `secmod.db` on Windows platforms and `secmodule.db` on Unix platforms). This database is stored in the same directory as your certificate database (`cert7.db`) and key database (`key3.db`), as indicated by the `-d` option of Netscape Signing Tool.

Before using Netscape Signing Tool in FIPS-validated mode, you must use Navigator to switch to FIPS mode. For information on how to do this, see Operating Netscape Navigator in FIPS PUB-140-1 Compliant Mode on Netscape DevEdge.

After switching the Navigator cryptographic module to FIPS mode, you have two choices:

- Use the same security module database from Netscape Signing Tool (by specifying the same directory with the `-d` option).
- Make a copy of Communicator's security module database and place it in Netscape Signing Tool's database directory.

Verifying FIPS Mode

Use the `-M` option to verify that you are using the FIPS-140-1 module.

This Unix example shows that Netscape Signing Tool is using a non-FIPS module:

```
% signtool -d "c:\netscape\users\jsmith" -M
using certificate directory: c:\netscape\users\jsmith
Listing of PKCS11 modules
-----------------------------------------------
  1. Netscape Internal PKCS #11 Module
          (this module is internally loaded)
          slots: 2 slots attached
          status: loaded
    slot: Communicator Internal Cryptographic Services Version 4.0
   token: Communicator Generic Crypto Svcs
    slot: Communicator User Private Key and Certificate Services
   token: Communicator Certificate DB
-----------------------------------------------
```

This Unix example shows that Netscape Signing Tool is using a FIPS-140-1 module:

% signtool -d "c:\netscape\users\jsmith" -M
using certificate directory: c:\netscape\users\jsmith
Enter Password or Pin for "Communicator Certificate DB": [password will not echo]
Listing of PKCS11 modules
----
  1. Netscape Internal FIPS PKCS #11 Module
          (this module is internally loaded)
          slots: 1 slots attached
          status: loaded
    slot: Netscape Internal FIPS-140-1 Cryptographic Services
   token: Communicator Certificate DB

Answers to Common Questions

This section answers the most common technical questions regarding Netscape Signing Tool.

Netscape Signing Tool, or Communicator, fails to report the presence of a particular certificate in the database, even though that certificate should be there.

Netscape Signing Tool 1.x and Communicator 4.x are designed to read only the `cert7.db` files used by Communicator 4.x. If it happens that a certificate gets downloaded with Navigator 3.x *after* Communicator 4.x was installed and run, that certificate is recorded in a database of the older (3.x) format. While Communicator does automatically convert Navigator's `cert5.db` and `key.db` databases to the `cert7.db` and `key3.db` formats the first time it runs, it does not do so again.

To get a certificate into the new database from an old one requires forcing Communicator to reinitialize its `cert7.db` file as it does at first run-time. This requires that the certificates in the current `cert7.db` file be exported for later re-importing.

1. Click the Security Info button on the Communicator toolbar.
2. Click Yours under Certificates in the left panel, and select a certificate to export.
3. Click Export and save a PKCS #12 copy of the certificate to a safe location (if no copy already exists).
4. Repeat steps 2 and 3 for each certificate present.
5. Exit Communicator completely.
6. Move the `cert7.db` and `key3.db` files from your user profile directory to a backup directory. This is a safety measure: these files shouldn't be needed again. Once the following steps are successfully completed, and you have used `signtool -l` to verify that the upgraded `cert7.db` file has the right certificates, you can discard these backup copies.
7. Copy Navigator's `cert5.db` and `key.db` files to your Communicator user-profile directory.
8. Restart Communicator. It automatically upgrades the older database files, including their contents.
9. Click the Security Info button on the Communicator toolbar, then click Yours under Certificates in the left panel.

10. Click Import a Certificate and give the database password.

11. Select a certificate file to open and give the certificate's password.

12. Repeat steps 9 and 10 for each certificate to be re-imported.

The certificate needed to sign an object is in the certificate database, but Netscape Signing Tool's `-l` and `-k` options report "Unable to find issuer certificate" or "Unknown user" errors.

Netscape Signing Tool 1.x reads the `cert7.db` files used by Communicator 4.x. Normally, `cert7.db` files record a certificate's complete certificate chain information using the PKCS #7 cryptographic message-syntax standard. However, Navigator 3.x wasn't designed to properly use this standard (it wasn't in wide use yet). Therefore, certificates used by Communicator 4.x that were originally imported with Navigator 3.x may not have all the certificate chain information required for object signing.

In the case of VeriSign Class 2 or Class 3 certificates, the missing portion of the chain is the intermediate node, since the root is provided with Communicator by default and the leaf is included in the existing certificate information. To get the intermediate portion, use Communicator 4.x and click these links for VeriSign Class 2 or VeriSign Class 3 certificate authority updates.

When trying to read a JAR file into Communicator 4.05 the error message "Inconsistent files in manifest" appears in the Java console.

Communicator 4.05 (and only that version) has a bug that makes it sensitive to the case of the JAR manifest's filename as stored in the META-INF directory. Version 4.05 requires the filename to be all lowercase. Although the JAR specification calls for case insensitivity and Netscape Signing Tool does generate a lowercase filename, an uppercase filename can appear if another tool is used to create the JAR, or case translation occurs on the Windows platform.

This problem can be repaired by re-signing the JAR with Netscape Signing Tool, or by unzipping the file and changing `MANIFEST.MF` to `manifest.mf`.

How can users change the nicknames of their own certificates?

For convenience, users may want to shorten the nicknames of some of the certificates they use. While certificates cannot be renamed directly, it may be possible to replace them and give the replacement a new name.

Note that this process can present a risk because it requires the user to delete a certificate before replacing it, and if the replacement fails and there is no backup certificate, the certificate is lost.

1. Click the Security Info button on Communicator's toolbar.
2. Click Yours under Certificates and select a certificate to rename.
3. Click Export and save a PKCS #12 copy of the certificate to a safe location (if no copy already exists). This copy is needed if replacement fails.
4. Click Delete and remove the certificate from the certificate database. Note that the certificate's corresponding private key isn't deleted, just the certificate itself.
5. Retrieve the certificate again from the Certificate Authority. The specific procedure for doing this depends on the Certificate Authority being used. Be very sure that the replacement is the correct one.
6. Enter a new name for the certificate when downloading it.

Note: The Export and Import buttons in the Security Info dialog box can't be used to change certificate nicknames. These functions can affect only a certificate's exported PKCS #12 filename.

When I click the Security icon in the Communicator toolbar, click Yours under Certificates, select my object-signing certificate, and click Verify, Communicator informs me that the certificate is not valid. Why?

This is a common occurrence. The Verify button works with S/MIME certificates only. It does not work with object-signing certificates.

To verify an object-signing certificate, use

`signtool -l -k` *nickname*

where *nickname* is the nickname of the certificate you want to verify.

I get the following error when trying to create a test certificate:

```
failure authenticating to key database .: Security I/O error.
```

This error typically means that you have not yet set a password for the certificate database. To set the password for the Communicator database, click the Security icon in the Communicator toolbar, click Passwords, and click Set Password to create a password.

Objects to be signed will be stored and used long-term, well after the certificates used for signing have expired. Will signed objects still be trusted even after their object-signing certificates have expired?

Although certificates expire, valid signatures do not. Signature validation is based on the date of the signature rather than the time verification occurs. If a certificate chain was valid at signing, Communicator will continue to recognize that signature even after certificates in that chain expire. Note that this would not be true, however, if an object was signed using the `-z` option which omits the original timestamp and forces validation to rely on the current status of the certificate chain.

Appendix

G

SSL Strength Tool

SSL Strength Tool is a command-line tool that connects to an SSL server and reports back the encryption cipher and strength used for the connection.

This appendix has the following sections:

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Syntax

```
sslstrength hostname[:port]
            [ciphers=ciphercode(s)]
```

```
[verbose]
[policy=export|domestic]
```

This form of the command returns a list of enabled ciphers on the client, then attempts to connect to the named SSL host, on the specified port. If the connection is successful, it returns information about the negotiated encryption strength.

```
sslstrength ciphers
```

This form of the command returns a list of the possible ciphers. A letter in the first column of the output is the code used by the `ciphers=` option. Pass any number of cipher codes to the `ciphers=` argument to identify the cipher preferences.

Options and Arguments

The SSL Strength Tool command options and their arguments are defined as follows:

Options and Arguments	
`hostname`	Required. Identifies the SSL server to which to connect.
`port`	Optional. Identifies a specific port on the specified SSL server to which to connect. If not specified, defaults to the standard HTTPS port, 443.
`ciphers=`	Optional. Turns on the cipher preferences corresponding to the specified cipher codes, and turns off all other cipher preferences. To obtain the list of cipher character codes, execute the special form of the command: `sslstrength ciphers`.

`verbose`	Optional. Turns on the verbose form of command output, which provides additional information about the progress of the connection.
`policy=`	Optional. Sets your policy regarding which ciphers can be permitted. Restricts the available ciphers to the same set used by Netscape Communicator for domestic or export versions (to comply with federal export restrictions). The value can be `export` or `domestic`. If not specified, defaults to `domestic`.

Usage

During an SSL handshake, the client sends the server a list of the ciphers it can use. The server chooses one of the ciphers based on its cipher policies, and notifies the client of which one to use.

When you issue the `sslstrength` command, the tool first prints the list of ciphers enabled on the client. It then connects to an SSL server and reports back the following information:

- The bulk encryption algorithm selected
- The key size selected
- The secret key size
- Information about the SSL server certificate, including:
  - The issuer subject name
  - The certificate subject name
  - The validity period

Restricting Ciphers

You can selectively enable or disable specific ciphers on the client, to determine what strength of connection is used for those ciphers. Use the `policy=` or `ciphers=` option to restrict which ciphers are available.

- To restrict the available ciphers to the same set used by Communicator for exportable or domestic versions, set the `policy=` option to either `domestic` or `export`. In an exportable client, only those ciphers that are valid for export are enabled.
- To further restrict the ciphers available, use the `ciphers=` option. The argument to this option is a string of characters, where each single character represents a cipher. For example, `ciphers=bfi` turns on the cipher preferences corresponding to the codes `b`, `f` ,and `i`. It turns off all other cipher preferences.

  To obtain the list of cipher character codes, execute this command:

  ```
  sslstrength ciphers
  ```

Export Policy and Step-up

Some institutions, such as banks, may be qualified to obtain a special "step-up" certificate (also know as a "global server ID") that allows the server to override export policy. When this certificate is installed in the server, it allows an export client that has step-up capability to renegotiate the SSL cipher and use domestic-strength encryption.

A connection that steps up starts out with 40-bit encryption, then, upon encountering a `change-cipher-spec` handshake, changes to 128-bit encryption. To check whether a client has stepped up correctly upon encountering a step-up certificate, check that it is using export policy, and that the secret key size is 128 bits.

Examples

The following examples show the output from various `sslstrength` commands.

Example 1

This example shows output from a command that allows all options to default.

```
sslstrength  myhost.netscape.com


Using domestic policy
Connecting to myhost.netscape.com:443
Using all ciphersuites usually found in client
Your Cipher preference:
id    CipherName                                        Domestic     Export
a     SSL_EN_RC4_128_WITH_MD5              (ssl2)    Yes          No
b     SSL_EN_RC2_128_CBC_WITH_MD5          (ssl2)    Yes          No
c     SSL_EN_DES_192_EDE3_CBC_WITH_MD5     (ssl2)    Yes          No
d     SSL_EN_DES_64_CBC_WITH_MD5           (ssl2)    Yes          No
e     SSL_EN_RC4_128_EXPORT40_WITH_MD5     (ssl2)    Yes          Yes
f     SSL_EN_RC2_128_CBC_EXPORT40_WITH_MD5 (ssl2)    Yes          Yes
i     SSL_RSA_WITH_RC4_128_MD5             (ssl3)    Yes Step-up only
j     SSL_RSA_WITH_3DES_EDE_CBC_SHA        (ssl3)    Yes Step-up only
k     SSL_RSA_WITH_DES_CBC_SHA             (ssl3)    Yes          No
l     SSL_RSA_EXPORT_WITH_RC4_40_MD5       (ssl3)    Yes          Yes
m     SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5   (ssl3)    Yes          Yes
o     SSL_RSA_WITH_NULL_MD5                (ssl3)    Yes          Yes
p     SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA   (ssl3)    Yes          No
q     SSL_RSA_FIPS_WITH_DES_CBC_SHA        (ssl3)    Yes          No
SSL Connection Status
Cipher:          RC4
Key Size:        128
Secret Key Size: 128
Issuer:          OU=Secure Server Certification Authority, O="RSA Data
Security, Inc.", C=US
Subject:         CN=myhost.netscape.com, OU=E-Store Merchant Server,
O=Netscape Communications Corp., L=Mountain View, ST=California, C=US
Valid:           from Fri Oct 02, 1998 to Sat Oct 02, 1999
```

Example 2

This example shows output from a command that limits the client to three ciphers.

```
sslstrength  myhost.netscape.com  ciphers=jkl


Using domestic policy
Connecting to myhost.netscape.com:443
Your Cipher preference:
```

```
id    CipherName                                        Domestic     Export
j     SSL_RSA_WITH_3DES_EDE_CBC_SHA        (ssl3)    Yes Step-up only
k     SSL_RSA_WITH_DES_CBC_SHA             (ssl3)    Yes          No
l     SSL_RSA_EXPORT_WITH_RC4_40_MD5       (ssl3)    Yes          Yes
SSL Connection Status
Cipher:          3DES-EDE-CBC
Key Size:        168
Secret Key Size: 168
Issuer:          OU=Secure Server Certification Authority, O="RSA Data
Security, Inc.", C=US
Subject:         CN=myhost.netscape.com, OU=E-Store Merchant Server,
O=Netscape Communications Corp., L=Mountain View, ST=California, C=US
Valid:           from Fri Oct 02, 1998 to Sat Oct 02, 1999
```

Example 3

This example shows output from a command that sets the client's policy to enable standard export ciphers.

```
sslstrength  myhost.netscape.com  policy=export


Using export policy
Connecting to myhost.netscape.com:443
Using all ciphersuites usually found in client
Your Cipher preference:
id    CipherName                                        Domestic     Export
e     SSL_EN_RC4_128_EXPORT40_WITH_MD5     (ssl2)    Yes          Yes
f     SSL_EN_RC2_128_CBC_EXPORT40_WITH_MD5 (ssl2)    Yes          Yes
i     SSL_RSA_WITH_RC4_128_MD5             (ssl3)    Yes Step-up only
j     SSL_RSA_WITH_3DES_EDE_CBC_SHA        (ssl3)    Yes Step-up only
l     SSL_RSA_EXPORT_WITH_RC4_40_MD5       (ssl3)    Yes          Yes
m     SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5   (ssl3)    Yes          Yes
o     SSL_RSA_WITH_NULL_MD5                (ssl3)    Yes          Yes
SSL Connection Status
Cipher:          RC4-40
Key Size:        128
Secret Key Size: 40
Issuer:          OU=Secure Server Certification Authority, O="RSA Data
Security, Inc.", C=US
Subject:         CN=myhost.netscape.com, OU=E-Store Merchant Server,
O=Netscape Communications Corp., L=Mountain View, ST=California, C=US
Valid:           from Fri Oct 02, 1998 to Sat Oct 02, 1999
```

Appendix

H

SSL Debugging Tool

SSL Debugging Tool is an SSL-aware command-line proxy. It watches TCP connections and displays the data going by. If a connection is SSL, the data display includes interpreted SSL records and handshaking information.

This appendix has the following sections:

Availability

This tool is available for AIX 4.3, OSF/1 v4.0D, Solaris 2.6 (SunOS 5.6), Solaris 8, and Windows NT 4.0.

Description

The `ssltap` command opens a socket on a rendezvous port and waits for an incoming connection from the client side. Once this connection arrives, the tool makes another connection to the specified host name and port on the server side. It passes any data sent by the client to the server and vice versa. The tool also displays the data to the shell window from which it was called. It can do this for plain HTTP connections or any TCP protocol, as well as for SSL streams, as described here.

The tool cannot and does not decrypt any encrypted message data. You use the tool to look at the plain text and binary data that are part of the handshake procedure, before the secure connection is established.

Syntax

To run SSL Debugging Tool, type this command in a command shell:

`ssltap` [`-vhfsxl`] [`-p` *port*] *hostname*:*port*

Options

The command does not require any options other than *hostname:port*, but you normally use them to control the connection interception and output. The options for the command are the following:

`-v`	Print a version string for the tool.
`-h`	Turn on hex/ASCII printing. Instead of outputting raw data, the command interprets each record as a numbered line of hex values, followed by the same data as ASCII characters. The two parts are separated by a vertical bar. Nonprinting characters are replaced by dots.
`-f`	Turn on fancy printing. Output is printed in colored HTML. Data sent from the client to the server is in blue; the server's reply is in red. When used with looping mode, the different connections are separated with horizontal lines. You can use this option to upload the output into a browser.

`-s`	Turn on SSL parsing and decoding. The tool does not automatically detect SSL sessions. If you are intercepting an SSL connection, use this option so that the tool can detect and decode SSL structures. If the tool detects a certificate chain, it saves the DER-encoded certificates into files in the current directory. The files are named `cert.0`*x*, where *x* is the sequence number of the certificate. If the `-s` option is used with `-h`, two separate parts are printed for each record: the plain hex/ASCII output, and the parsed SSL output.
`-x`	Turn on hex/ASCII printing of undecoded data inside parsed SSL records. Used only with the `-s` option. This option uses the same output format as the `-h` option.
`-l`	Turn on looping; that is, continue to accept connections rather than stopping after the first connection is complete.
`-p` *port*	Change the default rendezvous port (1924) to another port. The following are well-known port numbers: HTTP 80 HTTPS 443 SMTP 25 FTP 21 IMAP 143 IMAPS 993 (IMAP over SSL) NNTP 119 NNTPS 563 (NNTP over SSL)

Examples

You can use SSL Debugging Tool to intercept any connection information. Although you can run the tool at its most basic by issuing the `ssltap` command with no options other than *hostname:port*, the information you get in this way is not very useful.

For example, assume your development machine is called `intercept`. The simplest way to use the debugging tool is to execute the following command from a command shell:

```
ssltap www.netscape.com:80
```

The program waits for an incoming connection on the default port 1924. In your browser window, enter the URL `http://intercept:1924`. The browser retrieves the requested page from the server at `www.netscape.com`, but the page is intercepted and passed on to the browser by the debugging tool on `intercept`.

On its way to the browser, the data is printed to the command shell from which you issued the command. Data sent from the client to the server is surrounded by the following symbols:

`--> [` *data* `]`

Data sent from the server to the client is surrounded by the following symbols:

`<-- [` *data* `]`

The raw data stream is sent to standard output and is not interpreted in any way. This can result in peculiar effects, such as sounds, flashes, and even crashes of the command shell window. To output a basic, printable interpretation of the data, use the `-h` option, or, if you are looking at an SSL connection, the `-s` option.

You will notice that the page you retrieved looks incomplete in the browser. This is because, by default, the tool closes down after the first connection is complete, so the browser is not able to load images. To make the tool continue to accept connections, switch on looping mode with the `-l` option.

The following examples show the output from commonly used combinations of options.

Example 1

The `s` and `x` options in this example turn on SSL parsing and show undecoded values in hex/ASCII format. The output is routed to a text file.

Command

```
ssltap.exe -sx -p 444 interzone.mcom.com:443 > sx.txt
```

Output

```
Connected to interzone.mcom.com:443
```

```
--> [
alloclen = 66 bytes
   [ssl2]  ClientHelloV2 {
          version = {0x03, 0x00}
          cipher-specs-length = 39 (0x27)
          sid-length = 0 (0x00)
          challenge-length = 16 (0x10)
          cipher-suites = {
                (0x010080) SSL2/RSA/RC4-128/MD5
               (0x020080) SSL2/RSA/RC4-40/MD5
               (0x030080) SSL2/RSA/RC2CBC128/MD5
               (0x040080) SSL2/RSA/RC2CBC40/MD5
               (0x060040) SSL2/RSA/DES64CBC/MD5
               (0x0700c0) SSL2/RSA/3DES192EDE-CBC/MD5
               (0x000004) SSL3/RSA/RC4-128/MD5
               (0x00ffe0) SSL3/RSA-FIPS/3DES192EDE-CBC/SHA
               (0x00000a) SSL3/RSA/3DES192EDE-CBC/SHA
               (0x00ffe1) SSL3/RSA-FIPS/DES64CBC/SHA
               (0x000009) SSL3/RSA/DES64CBC/SHA
               (0x000003) SSL3/RSA/RC4-40/MD5
               (0x000006) SSL3/RSA/RC2CBC40/MD5
               }
          session-id = { }
          challenge = { 0xec5d 0x8edb 0x37c9 0xb5c9 0x7b70 0x8fe9 0xd1d3
0x2592 }
}
]
<-- [
SSLRecord {
   0: 16 03 00 03  e5                                      |.....
   type    = 22 (handshake)
   version = { 3,0 }
   length  = 997 (0x3e5)
   handshake {
   0: 02 00 00 46                                         |...F
     type = 2 (server_hello)
     length = 70 (0x000046)
          ServerHello {
          server_version = {3, 0}
          random = {...}
   0: 77 8c 6e 26  6c 0c ec c0  d9 58 4f 47  d3 2d 01 45  |
wŒn&amp;l.ì..XOG.-.E
   10: 5c 17 75 43  a7 4c 88 c7  88 64 3c 50  41 48 4f 7f  |
\.uC§L.Ç.d&lt;PAHO.
               session ID = {
               length = 32
                contents = {..}
   0: 14 11 07 a8  2a 31 91 29  11 94 40 37  57 10 a7 32  |
...¨*1.)..@7W.§2
```

```
   10: 56 6f 52 62  fe 3d b3 65  b1 e4 13 0f  52 a3 c8 f6  |
VoRbþ=³e±...R£È.
      }
            cipher_suite = (0x0003) SSL3/RSA/RC4-40/MD5
      }
   0: 0b 00 02 c5                                          |...Å
     type = 11 (certificate)
     length = 709 (0x0002c5)
          CertificateChain {
          chainlength = 706 (0x02c2)
             Certificate {
          size = 703 (0x02bf)
             data = { saved in file 'cert.001' }
          }
      }
   0: 0c 00 00 ca                                          |....
      type = 12 (server_key_exchange)
      length = 202 (0x0000ca)
   0: 0e 00 00 00                                          |....
      type = 14 (server_hello_done)
      length = 0 (0x000000)
   }
}
]
--> [
SSLRecord {
   0: 16 03 00 00  44                                       |....D
   type    = 22 (handshake)
   version = { 3,0 }
   length  = 68 (0x44)
   handshake {
   0: 10 00 00 40                                          |...@
   type = 16 (client_key_exchange)
   length = 64 (0x000040)
      ClientKeyExchange {
         message = {...}
      }
   }
}
]
--> [
SSLRecord {
   0: 14 03 00 00  01                                       |.....
   type    = 20 (change_cipher_spec)
   version = { 3,0 }
   length  = 1 (0x1)
   0: 01                                                   |.
}
SSLRecord {
   0: 16 03 00 00  38                                       |....8
```

```
    type    = 22 (handshake)
    version = { 3,0 }
    length  = 56 (0x38)
              < encrypted >
}
]
<-- [
SSLRecord {
    0: 14 03 00 00  01                                      |.....
    type    = 20 (change_cipher_spec)
    version = { 3,0 }
    length  = 1 (0x1)
    0: 01                                                  |.
}
]
<-- [
SSLRecord {
    0: 16 03 00 00  38                                      |....8
    type    = 22 (handshake)
    version = { 3,0 }
    length  = 56 (0x38)
                < encrypted >
}
]
--> [
SSLRecord {
    0: 17 03 00 01  1f                                      |.....
    type    = 23 (application_data)
    version = { 3,0 }
    length  = 287 (0x11f)
              < encrypted >
}
]
<-- [
SSLRecord {
    0: 17 03 00 00  a0                                      |....
    type    = 23 (application_data)
    version = { 3,0 }
    length  = 160 (0xa0)
              < encrypted >
}
]
<-- [
SSLRecord {
0: 17 03 00 00  df                                     |....ß
    type    = 23 (application_data)
    version = { 3,0 }
    length  = 223 (0xdf)
              < encrypted >
```

```
}
SSLRecord {
   0: 15 03 00 00  12                                      |.....
   type    = 21 (alert)
   version = { 3,0 }
   length  = 18 (0x12)
             < encrypted >
}
]
Server socket closed.
```

Example 2

The `-s` option turns on SSL parsing. Because the `-x` option is not used in this example, undecoded values are output as raw data. The output is routed to a text file.

Command

```
ssltap.exe -s  -p 444 interzone.mcom.com:443 > s.txt
```

Output

```
Connected to interzone.mcom.com:443
--> [
alloclen = 63 bytes
   [ssl2]  ClientHelloV2 {
          version = {0x03, 0x00}
          cipher-specs-length = 36 (0x24)
          sid-length = 0 (0x00)
          challenge-length = 16 (0x10)
          cipher-suites = {
               (0x010080) SSL2/RSA/RC4-128/MD5
               (0x020080) SSL2/RSA/RC4-40/MD5
               (0x030080) SSL2/RSA/RC2CBC128/MD5
               (0x060040) SSL2/RSA/DES64CBC/MD5
               (0x0700c0) SSL2/RSA/3DES192EDE-CBC/MD5
               (0x000004) SSL3/RSA/RC4-128/MD5
               (0x00ffe0) SSL3/RSA-FIPS/3DES192EDE-CBC/SHA
               (0x00000a) SSL3/RSA/3DES192EDE-CBC/SHA
               (0x00ffe1) SSL3/RSA-FIPS/DES64CBC/SHA
               (0x000009) SSL3/RSA/DES64CBC/SHA
               (0x000003) SSL3/RSA/RC4-40/MD5
               }
             session-id = { }
          challenge = { 0x713c 0x9338 0x30e1 0xf8d6 0xb934 0x7351 0x200c
```

```
0x3fd0 }
]
<-- [
SSLRecord {
    type    = 22 (handshake)
    version = { 3,0 }
    length  = 997 (0x3e5)
    handshake {
        type = 2 (server_hello)
        length = 70 (0x000046)
            ServerHello {
            server_version = {3, 0}
            random = {...}
            session ID = {
                length = 32
                contents = {..}
                }
                cipher_suite = (0x0003) SSL3/RSA/RC4-40/MD5
            }
        type = 11 (certificate)
        length = 709 (0x0002c5)
            CertificateChain {
                chainlength = 706 (0x02c2)
                Certificate {
                  size = 703 (0x02bf)
                  data = { saved in file 'cert.001' }
                }
            }
        type = 12 (server_key_exchange)
        length = 202 (0x0000ca)
        type = 14 (server_hello_done)
        length = 0 (0x000000)
    }
}
]
--> [
SSLRecord {
    type    = 22 (handshake)
    version = { 3,0 }
    length  = 68 (0x44)
    handshake {
        type = 16 (client_key_exchange)
        length = 64 (0x000040)
            ClientKeyExchange {
                message = {...}
            }
    }
}
]
--> [
```

```
SSLRecord {
   type    = 20 (change_cipher_spec)
   version = { 3,0 }
   length  = 1 (0x1)
}
SSLRecord {
   type    = 22 (handshake)
   version = { 3,0 }
   length  = 56 (0x38)
             < encrypted >
}
]
<-- [
SSLRecord {
   type    = 20 (change_cipher_spec)
   version = { 3,0 }
   length  = 1 (0x1)
}
]
<-- [
SSLRecord {
   type    = 22 (handshake)
   version = { 3,0 }
   length  = 56 (0x38)
             < encrypted >
}
]
--> [
SSLRecord {
   type    = 23 (application_data)
   version = { 3,0 }
   length  = 287 (0x11f)
             < encrypted >
}
]
[
SSLRecord {
   type    = 23 (application_data)
   version = { 3,0 }
   length  = 160 (0xa0)
             < encrypted >
}
]
<-- [
SSLRecord {
   type    = 23 (application_data)
   version = { 3,0 }
   length  = 223 (0xdf)
             < encrypted >
}
```

```
SSLRecord {
    type    = 21 (alert)
    version = { 3,0 }
    length  = 18 (0x12)
              < encrypted >
}
]
Server socket closed.
```

Example 3

In this example, the `-h` option turns hex/ASCII format. There is no SSL parsing or decoding. The output is routed to a text file.

Command

```
ssltap.exe -h  -p 444 interzone.mcom.com:443 > h.txt
```

Output

```
Connected to interzone.mcom.com:443
--> [
    0: 80 40 01 03  00 00 27 00  00 00 10 01  00 80 02 00  |
.@....'.........
    10: 80 03 00 80  04 00 80 06  00 40 07 00  c0 00 00 04  |
.........@......
    20: 00 ff e0 00  00 0a 00 ff  e1 00 00 09  00 00 03 00  |
........á.......
    30: 00 06 9b fe  5b 56 96 49  1f 9f ca dd  d5 ba b9 52  |
..›þ[V.I.Ÿ...ºªR
    40: 6f 2d                                              |o-
]
<-- [
    0: 16 03 00 03  e5 02 00 00  46 03 00 7f  e5 0d 1b 1d  |
........F.......
    10: 68 7f 3a 79  60 d5 17 3c  1d 9c 96 b3  88 d2 69 3b  |
h.:y‘..&lt;.œ.³.Òi;
    20: 78 e2 4b 8b  a6 52 12 4b  46 e8 c2 20  14 11 89 05  | x.K.¦R.KFè.
..‰.
    30: 4d 52 91 fd  93 e0 51 48  91 90 08 96  c1 b6 76 77  |
MR.ý..QH.....¶vw
    40: 2a f4 00 08  a1 06 61 a2  64 1f 2e 9b  00 03 00 0b  |
*ô..¡.a¢d..›....
    50: 00 02 c5 00  02 c2 00 02  bf 30 82 02  bb 30 82 02  |
..Å......0...0..
    60: 24 a0 03 02  01 02 02 02  01 36 30 0d  06 09 2a 86  | $
```

```
.......60...*.
   70: 48 86 f7 0d  01 01 04 05  00 30 77 31  0b 30 09 06  |
H.÷......0w1.0..
   80: 03 55 04 06  13 02 55 53  31 2c 30 2a  06 03 55 04  |
.U....US1,0*..U.
   90: 0a 13 23 4e  65 74 73 63  61 70 65 20  43 6f 6d 6d  | ..#Netscape
Comm
   a0: 75 6e 69 63  61 74 69 6f  6e 73 20 43  6f 72 70 6f  | unications
Corpo
   b0: 72 61 74 69  6f 6e 31 11  30 0f 06 03  55 04 0b 13  |
ration1.0...U...
   c0: 08 48 61 72  64 63 6f 72  65 31 27 30  25 06 03 55  |
.Hardcore1'0%..U
   d0: 04 03 13 1e  48 61 72 64  63 6f 72 65  20 43 65 72  | ....Hardcore
Cer
   e0: 74 69 66 69  63 61 74 65  20 53 65 72  76 65 72 20  | tificate
Server
   f0: 49 49 30 1e  17 0d 39 38  30 35 31 36  30 31 30 33  |
II0...9805160103
<additional data lines>
]
<additional records in same format>
Server socket closed.
```

Example 4

In this example, the `-s` option turns on SSL parsing, and the `-h` options turns on hex/ASCII format. Both formats are shown for each record. The output is routed to a text file.

Command

```
ssltap.exe -hs -p 444 interzone.mcom.com:443 > hs.txt
```

Output

```
Connected to interzone.mcom.com:443
--> [
   0: 80 3d 01 03  00 00 24 00  00 00 10 01  00 80 02 00  |
.=....$.........
   10: 80 03 00 80  04 00 80 06  00 40 07 00  c0 00 00 04  |
.........@......
   20: 00 ff e0 00  00 0a 00 ff  e1 00 00 09  00 00 03 03  |
........á.......
   30: 55 e6 e4 99  79 c7 d7 2c  86 78 96 5d  b5 cf e9
|U..™yÇ×,.x.]µÏé
```

```
alloclen = 63 bytes
   [ssl2]  ClientHelloV2 {
          version = {0x03, 0x00}
          cipher-specs-length = 36 (0x24)
          sid-length = 0 (0x00)
          challenge-length = 16 (0x10)
          cipher-suites = {
               (0x010080) SSL2/RSA/RC4-128/MD5
               (0x020080) SSL2/RSA/RC4-40/MD5
               (0x030080) SSL2/RSA/RC2CBC128/MD5
               (0x040080) SSL2/RSA/RC2CBC40/MD5
               (0x060040) SSL2/RSA/DES64CBC/MD5
               (0x0700c0) SSL2/RSA/3DES192EDE-CBC/MD5
               (0x000004) SSL3/RSA/RC4-128/MD5
               (0x00ffe0) SSL3/RSA-FIPS/3DES192EDE-CBC/SHA
               (0x00000a) SSL3/RSA/3DES192EDE-CBC/SHA
               (0x00ffe1) SSL3/RSA-FIPS/DES64CBC/SHA
               (0x000009) SSL3/RSA/DES64CBC/SHA
               (0x000003) SSL3/RSA/RC4-40/MD5
               }
          session-id = { }
          challenge = { 0x0355 0xe6e4 0x9979 0xc7d7 0x2c86 0x7896 0x5db

0xcfe9 }
}
]
<additional records in same formats>
Server socket closed.
```

Usage Tips

- When SSL restarts a previous session, it makes use of cached information to do a partial handshake. If you wish to capture a full SSL handshake, restart the browser to clear the session id cache.
- If you run the tool on a machine other than the SSL server to which you are trying to connect, the browser will complain that the host name you are trying to connect to is different from the certificate. If you are using the default BadCert callback, you can still connect through a dialog. If you are not using the default BadCert callback, the one you supply must allow for this possibility.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W