[MS-CSRA]:
Certificate Services Remote Administration Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
12/18/2006	0.1		MCPP Milestone 2 Initial Availability
03/02/2007	1.0		MCPP Milestone 2
04/03/2007	1.1		Monthly release
05/11/2007	1.2		Monthly release

Date	Revision History	Revision Class	Comments
06/01/2007	2.0	Major	Updated and revised the technical content.
07/03/2007	2.1	Minor	Updates for minor issues.
07/20/2007	2.2	Minor	Updates for minor issues.
08/10/2007	2.2.1	Editorial	Revised and edited the technical content.
09/28/2007	2.3	Minor	Updated the technical content.
10/23/2007	3.0	Major	Updated and revised the technical content.
11/30/2007	4.0	Major	Updated and revised the technical content.
01/25/2008	5.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction

This document specifies the Certificate Services Remote Administration Protocol. The protocol consists of a set of **Distributed Component Object Model (DCOM)** interfaces, as specified in [MS-DCOM], that allow administrative tools to configure the state and policy of a **certification authority (CA)** on a server.

Familiarity with **public key infrastructure (PKI)** concepts such as asymmetric and symmetric cryptography, asymmetric and **symmetric encryption** techniques, digital **certificate** concepts, and cryptographic key establishment are required for a complete understanding of this specification. In addition, a comprehensive understanding of the X509 standard, as specified in [X509], is required for a complete understanding of the protocol and its usage. The Handbook of Applied Cryptography provides an excellent introduction to cryptography and PKI concepts. For more information, see [CRYPTO]. X509, as specified in [X509], provides an excellent introduction to PKI and certificate concepts. Certificate **Revocation** and Status Checking provides an excellent introduction to **certificate revocation lists (CRL)** and revocation concepts. For more information, see [MSFT-CRL].

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Access Control Entry (ACE)
Access Control List (ACL)
Active Directory (AD)
Attribute
Certificate
Certificate Authority (CA)
Certificate Revocation Lists (CRL)
Certificate Services
Certificate Template
Client
Common Name (CN)
Container
Cryptographic Service Provider (CSP)
Discretionary Access Control List (DACL)
Distinguished Name (DN)
Distributed Component Object Model (DCOM)
Domain
Domain Controller (DC)
Encryption
Enrollment
Enterprise CA
Exchange Certificate
Forest
Key Archival
Key Distribution Center (KDC)
Key Escrow
Key Recovery Agent (KRA)
Key Recovery Certificate
Lightweight Directory Access Protocol (LDAP)
Object
Object Identifier (OID)
Offline
Principal

Private Key
Public Key
Public Key Algorithm
Public Key Infrastructure (PKI)
Public-Private Key Pair
Revocation
Role Separation
Root CA
Sanitized Name
Schema
Signing Certificates
Standalone CA
Symmetric Algorithm
Symmetric Encryption
Universal Naming Convention (UNC)

The following terms are specific to this document:

Asymmetric Algorithm: A synonym for **public key algorithm**. For more information, including an introduction to these concepts and terminology, see [PUBKEY] and [RSAFAQ].

See also **Public Key Algorithm**.

Certificate Authority (CA) Roles: A list of administrator-defined rights or **access control lists (ACLs)** that define the capability of a given **principal** on a **certificate authority (CA)**. CA Roles are specified in [CIMC-PP] section 5.2 and include administrator, operator, officer, and auditor.

Enrollment Agent Rights: A list of administrator-defined rights or **access control lists (ACLs)** that define the capability of a given **principal** to obtain a **certificate**, with subject information pertaining to a different **principal**, from a **certificate authority (CA)**. Enrollment Agent is not one of the roles defined in [CIMC-PP].

Index: A data structure used for locating data quickly within a **table**. For more information, see [GRAY].

Issuance: See **Certificate**.

Log Files: A representation of the history of Windows behavior: Windows Server 2003 stores request submissions and **certificate revocation** that has occurred since the last **log file** truncation or backup. **Log file** volume increases as database activity occurs. The **log files** can be decreased in size by performing a backup and then calling BackupTruncateLogs (as specified in section 2.2.2.1) of data value and structure changes in a database, stored in stable storage and used by the database to restore the last committed values of data items. For more information, see [GRAY].<1>

Officer Rights: A list of administrator-defined rights or **access control lists (ACLs)** that define the capability of a given officer (one of the roles specified in [CIMC-PP]) to approve certificate requests associated with a given set of principals. **Officer rights**, as specified in [CIMC-PP], are locally configured and stored on a **CA** and MUST be enforced by the **CA**.

Policy Module: A software module invoked by a **certificate authority (CA)** for the purpose of performing policy enforcement as part of certificate request processing. These modules are often extensions provided by an administrator or third party to a **certificate authority (CA)**. Policy modules may be chained together. Not all **certificate authorities (CAs)**) support policy modules.

Social Engineering: The class of attacks in which the attacker uses human-to-human interactions to gain user rights improperly.

Subordinate Certificate Authority (CA): A type of **CA** that is not a **root CA** for a relying party or **client**. A **subordinate CA** is a **CA** whose **certificate** is signed by some other **CA**, as specified in [RFC2510].

Table: A set of data elements that is organized into a predefined format of rows and columns. For more information, see [GRAY].

Unrevoke: Using the **RevokeCertificate** method, to change the status of a certificate with **Request.Disposition** "certificate revoked" and **Request.Revoked.Reason** "certificateHold" to **Request.Disposition** "certificate issued". As detailed in this document in the server processing rules for the RevokeCertificate method, only a certificate with **Request.Disposition** set to "certificate revoked" and **Request.Revoked.Reason** set to "certificateHold" can be unrevoked.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[CIMC-PP] National Security Agency, "National Information Assurance Partnership", http://www.nsa.gov/ia/industry/niap.cfm

[ITUX690] ITU-T, "ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Recommendation X.690, July 2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf

[MS-ADA1] Microsoft Corporation, "Active Directory Schema Attributes A-L", June 2007.

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes", June 2007.

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", June 2007.

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

[MS-DOCO] Microsoft Corporation, "Windows Protocols Documentation Roadmap", January 2007.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions", January 2007.

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol Specification", June 2007.

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol Specification", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-WCCE] Microsoft Corporation, "Windows Client Certificate Enrollment Protocol Specification", June 2007.

[MSFT-TEMPLATES] Microsoft Corporation, "Implementing and Administering Certificate Templates in Windows Server 2003", July 2004, http://technet2.microsoft.com/WindowsServer/en/library/c25f57b0-5459-4c17-bb3f-2f657bd23f781033.mspx

If you have any trouble finding [MSFT-TEMPLATES], please check here.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2251] Wahl, M., Howes, T., and Kille, S., "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997, http://www.ietf.org/rfc/rfc2251.txt

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, March 1998, http://www.ietf.org/rfc/rfc2315.txt

[RFC2478] Baize, E. and Pinkas, D., "The Simple and Protected GSS-API Negotiation Mechanism", RFC 2478, December 1998, http://www.ietf.org/rfc/rfc2478.txt

[RFC2510] Adams, C. and Farrell, S., "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999, http://www.ietf.org/rfc/rfc2510.txt

[RFC2559] Boeyen, S., Howes, T., and Richard, P., "Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2", RFC 2559, April 1999, http://www.ietf.org/rfc/rfc2559.txt

[RFC2797] Myers, M., Liu, X., Schaad, J., and Weinstein, J., "Certificate Management Messages Over CMS", RFC 2797, April 2000, http://www.ietf.org/rfc/rfc2797.txt

[RFC2986] Nystrom, M. and Kaliski, B., "PKCS#10: Certificate Request Syntax Specification", RFC 2986, November 2000, http://www.ietf.org/rfc/rfc2986.txt

[RFC3280] Housley, R., Polk, W., Ford, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002, http://www.ietf.org/rfc/rfc3280.txt

[RFC4120] Neuman, C., Yu, T., Hartman, S., and Raeburn, K., "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005, http://www.ietf.org/rfc/rfc4120.txt

[RFC4523] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates", RFC 4523, June 2006, http://www.ietf.org/rfc/rfc4523.txt

[X509] ITU-T, "Information Technology - Open Systems Interconnection - The Directory: Public-Key and Attribute Certificate Frameworks", Recommendation X.509, August 2005, http://www.itu.int/rec/T-REC-X.509/en

Note There is a charge to download the specification.

[X660] ITU-T, "Information Technology - Open Systems Interconnection - Procedures for the Operation of OSI Registration Authorities: General Procedures and Top Arcs of the ASN.1 Object Identifier Tree", Recommendation X.660, August 2004, http://www.itu.int/rec/T-REC-X.660/en

Note There is a charge to download the specification.

[X680] ITU-T, "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", Recommendation X.680, July 2002, http://www.itu.int/rec/T-REC-X.680/en

Note There is a charge to download the specification.

[X690] ITU-T, "Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Recommendation X.690, July 2002, http://www.itu.int/rec/T-REC-X.690/en

Note There is a charge to download the specification.

1.2.2 Informative References

[CRYPTO] Menezes, A., Vanstone, S., and Oorschot, P., "Handbook of Applied Cryptography", 1997, http://www.cacr.math.uwaterloo.ca/hac/

[GRAY] Gray, G. and A. Reuter, "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[MSDN-CSPR] Microsoft Corporation, "Cryptographic Service Providers", http://msdn2.microsoft.com/en-us/library/aa380245.aspx

[MSFT-ARCHIVE] Microsoft Corporation, "Key Archival and Management in Windows Server 2003", December 2004, http://technet2.microsoft.com/WindowsServer/en/Library/296f87df-06c3-4e27-89ff-5 283cb76fb811033.mspx

[MSFT-CRL] Microsoft Corporation, "Certificate Revocation and Status Checking", January 2006, http://www.microsoft.com/technet/prodtechnol/winxppro/support/tshtcrl.mspx

[MSFT-PKI] Microsoft Corporation, "Best Practices for Implementing a Microsoft Windows Server 2003 Public Key Infrastructure", July 2004, http://technet2.microsoft.com/WindowsServer/en/library/091cda67-79ec-481d-8a96-03e0be7374ed1033.mspx

[PUBKEY] RSA Laboratories, "Crypto FAQ: Chapter 2 Cryptography: 2.1 Cryptographic Tools: 2.1.1 What Is Public-Key Cryptography?", http://www.rsa.com/rsalabs/node.asp?id=2165

[RSAFAQ] RSA Laboratories, "Frequently Asked Questions About Today's Cryptography, Version 4.1", May 2000, http://www.rsa.com/rsalabs/faq/files/rsalabs_faq41.pdf

1.3 Protocol Overview (Synopsis)

The Certificate Services Remote Administration Protocol consists of a set of Distributed Component Object Model (DCOM) interfaces, as specified in [MS-DCOM], that allow administrative tools to configure the state and policy of a certification authority (CA) on a server. The administrative tools may perform such functions as getting or setting properties on a CA, retrieving data, revoking certificates, or retrieving escrowed **private keys** from a CA.

Figure 1 reflects only CA administration, not the normal operation of the CA. The protocol for the normal operation of the Microsoft CA is specified in [MS-WCCE].

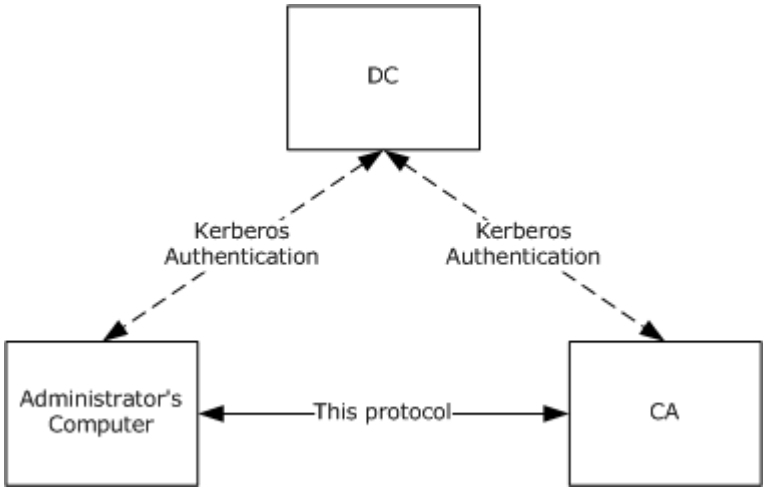


Figure 1: Machines involved in remote administration

In Figure 1, the principal components are:

- CA: The certification authority (CA) that receives configuration and administration tasks. The remote administration protocol defined in this document covers the interactions shown as a bold line in Figure 1.
- Administrator's Computer: A **client** to the CA that performs remote configuration or administration tasks.
- Domain Controller (DC): In most cases, a Kerberos **Key Distribution Center (KDC)** used to authenticate the parties for authenticated DCOM messages. The protocol documented here is built on top of authenticated DCOM messages. Interactions with the **DC** are shown in Figure 1 as dashed lines. DCOM is documented as specified in [MS-DCOM], which in turn references interactions with the domain controller.<2>

The protocol uses two DCOM interfaces: ICertAdminD (section 3.2.5.1) and ICertAdminD2 (section 3.2.5.2), which offer additional methods. Between the two, 48 methods are defined.

The methods of this protocol fall into the following categories:

- Managing pending certificate requests: A certificate request can be fulfilled immediately or can be held for human administrator approval or other action. When a request is pending human approval, there are ICertAdminD methods that allow the human's administrative console to interact with the CA to query and modify pending requests, as specified in [MS-WCCE] section 2.2.2).
- Configuring or retrieving data from CA databases: For purposes of this protocol, a CA must be built around a logical database, as specified in section 1.3.1.3. A number of methods in this protocol deal with configuration or data retrieval of particular rows or columns of **tables** in the logical database.
- Managing revocation: This protocol includes methods to tell the CA to revoke a certificate, to query the validity of a certificate, and to deal with the mechanics of publication of certificate revocation lists (CRLs).
- Managing audit: This protocol includes methods that allow the administrator to learn and specify which classes of events generate audit trail entries.

- Archived key retrieval: This protocol defines one method for retrieving a private key that was archived as part of a certificate request.
- Miscellaneous administrative actions: This protocol includes a number of methods for miscellaneous administrative actions—such as determining if the CA is responsive, determining what kinds of rights the caller has, telling the CA to go **offline**, and querying and editing various CA state variables. For details, see the descriptions in sections 3.2.5.1 and 3.2.5.2.

1.3.1 Concepts

The sections that follow define concepts and technologies used by the Certificate Services Remote Administration Protocol.

1.3.1.1 Number Annotation

Numbers expressed in the format 0xXXXX are to be interpreted as hexadecimal. Otherwise, all numbers are to be interpreted as decimal.

1.3.1.2 Object Identifiers

The protocol uses **object identifier (OID)** as unique identifiers for several classes of **objects**, as specified in [X660] and [RFC3280] Appendix A. OIDs are used to uniquely identify **certificate templates** that are available to the CA. Within a certificate, OIDs are used to identify standard extensions, as specified in [RFC3280] section 4.2.1, and some nonstandard extensions.

1.3.1.3 CA Databases

The protocol refers to four databases as tables with rows and columns hosted by the CA. There are two main tables: one for requests and one for CRLs. The request table has two auxiliary tables: one for a list of **attributes** for a particular request and one for a list of extensions for a particular request.

The four tables include:

- Request table: The request table holds the history of all requests to the CA, both completed and pending, one row per request.
- Attribute table: The attribute table holds the attributes, as specified in [RFC2986], that are contained within a specified certificate request.
- Extension table: The extension table holds the X.509 extensions, as specified in [X509], that are contained within a specified certificate request.
- CRL table: The CRL table holds the revocation data and status for the CA. The CA maintains a CRL database in the form of a table that holds all CRLs (both base and delta, as defined in [RFC3280] section 5), that have been issued.

Methods of this protocol refer to the above tables, which are specified in section 3.2.1.

1.3.1.4 CA Roles and Officer Rights

This protocol includes methods to get and set **CA roles** and **officer rights** (as specified in sections 3.2.5.2.6, 3.2.5.2.7, 3.2.5.2.12, and 3.2.5.2.13). CA roles are as specified in [CIMC-PP] section 5.2 and include administrator, operator, officer, and auditor. In addition, this protocol contains methods to assign **Enrollment Agent rights** on the CA. While "Enrollment Agent" can be considered a role, it is not one of the CA roles specified in [CIMC-PP].

1.3.1.5 Certificate Templates

An **enterprise CA** MUST use certificate templates configured locally to support certificate **enrollment** requests, as specified in [MS-WCCE]. The complete definition of certificate templates, including the list of attributes, flags, and extensions that have been implemented in Windows Server 2003 and Windows 2000,is specified in [MS-CRTD]. <3>

1.3.1.6 Sanitizing Common Names

The **common names (CN)** of the **Active Directory (AD)** objects, as specified in [MS-ADTS], that are used by the enrollment protocol are created by sanitizing the names of other objects and shortening the **sanitized name** so that it does not exceed 64 characters including spaces. Objects are defined as a collection of **LDAP** attributes. Attributes are defined as LDAP data types, as specified in [RFC2251] and [RFC4523]. The sanitized name MUST not exceed 64 characters (Bytes) in length. A name is sanitized by replacing disallowed characters with an exclamation point character (!) followed by four hexadecimal digits making one value that represents the 16-bit character being replaced.

In the following example, the parenthesis character (() is replaced with !0028, the number sign character (#) is replaced by !0023, the percent character (%) is replaced by !0025 and the "^" character is replaced by !005e.

```
Original Name:
     'LongCAName(WithSpeci@#$%^Characters'
Sanitized Name:
     'LongCAName!0028WithSpeci@!0023$!0025!005eCharacters'
```

The algorithm for creating a sanitized name is specified in section 3.3.1.

1.4 Relationship to Other Protocols

The Certificate Services Remote Administration Protocol depends on the DCOM Remote protocol, as specified in [MS-DCOM]. Microsoft DCOM negotiates its authentication method using GSS API, as specified in [RFC2478]. Either NTLM, as specified in [MS-NLMP], or Kerberos, as specified in [RFC4120] and [MS-KILE], may be selected as the authentication method.

No other Windows protocol directly depends on the Certificate Services Remote Administration Protocol. This protocol is designed to manage a server that implements the Windows Client Certificate Enrollment Protocol, as specified in [MS-WCCE].

1.5 Prerequisites/Preconditions

1.5.1 Certificate Template

The Certificate Services Remote Administration Protocol enables the configuration, setting of properties, or retrieval of properties on a CA. A CA can use templates in support of the Certificate Services Enrollment Protocol, as specified in [MS-WCCE]. An enterprise CA MUST have valid templates configured on the CA. Information on certificate templates can be found in [MSFT-TEMPLATES].

1.5.2 CA Name

The Certificate Services Remote Administration Protocol assumes the client knows the name of the CA server<4> that implements the DCOM interfaces specified in section 3.1.5.

1.6 Applicability Statement

This protocol provides clients with the capability to interact with CA for the purpose of managing X.509 certificates, as specified in [X509], or a CA configuration.

1.7 Versioning and Capability Negotiation

The Certificate Services Remote Administration Protocol is based on DCOM technology, as specified in [MS-DCOM], which provides capabilities to query for interface versions. Clients use the **IUnknown.QueryInterface** method to determine the supported server interface version.<5>

1.8 Vendor-Extensible Fields

For the Certificate Services Remote Administration Protocol, the CA MUST have the same concept of a CASigningList as specified in [MS-WCCE] section 3.2. In addition, the CA MUST maintain the information detailed as "Required" in section 3.2.1 of this protocol specification.

Vendors MAY implement the fields detailed as "Optional" in section 3.2.1 of this protocol specification, or any additional fields.

1.9 Standards Assignments

No standards assignments have been received for the Certificate Services Remote Administration Protocol described in this document.

2 Messages

The following sections specify how Certificate Services Remote Administration Protocol messages are transported, and Certificate Services Remote Administration Protocol message syntax.

2.1 Transport

DCOM, as specified in [MS-DCOM], is used as the transport protocol. The enrollment protocol documented here relies upon DCOM authentication and **encryption**, as specified in [MS-DCOM], for all protocol messages.<6>

2.2 Message Syntax

2.2.1 Common Structures

This section defines the structures used by the Certificate Services Remote Administration Protocol. These structures are used when performing various operations (using interface methods specified in section 3.2.5) on the server and as part of the server's response. This protocol shares a number of structures with the Windows Client Certificate Enrollment Protocol (as specified in [MS-WCCE]), which are specified in the following sections.

2.2.1.1 BYTE

The **BYTE** type specifies an 8-bit data item that corresponds to a single octet in a network protocol.

This type is declared as follows:

```
typedef byte BYTE;
```

2.2.1.2 VARIANT

The **VARIANT** type is implemented as specified in [MS-OAUT] section 2.2.26.

2.2.1.3 CERTVIEWRESTRICTION

The **CERTVIEWRESTRICTION** structure is used to restrict the data set returned by the CA server during calls to the **OpenView** method for the **ICertAdminD** interface.

This structure is passed by RPC, as specified in [MS-RPCE], and does not need special marshaling.

```
typedef struct _CERTVIEWRESTRICTION {
  DWORD ColumnIndex;
  LONG SeekOperator;
  LONG SortOrder;
  [size_is(cbValue), unique] BYTE* pbValue;
  DWORD cbValue;
} CERTVIEWRESTRICTION;
```

ColumnIndex: Unsigned integer value that specifies the identifier for the database column that is receiving the restriction.

SeekOperator: Integer value that specifies the logical operator of the data-query qualifier for the column. This parameter MUST be set to one of the following values:

Value	Meaning
0x00000001	Equal to
0x00000002	Less than
0x00000004	Less than or equal to
0x00000008	Greater than or equal to
0x00000010	Greater than

SortOrder: Integer value that specifies the sort order for the column. This parameter MUST be set to one of the following values:

Value	Meaning
0x00000000	No sort order
0x00000001	Ascending
0x00000002	Descending

pbValue: Pointer to a byte array that specifies the value against which the value in the corresponding column (specified by **ColumnIndex**) is compared, using SeekOperator.

cbValue: Unsigned integer value that specifies the length of the byte array pointed by **pbValue** field.

2.2.1.4 CERTTRANSBLOB

The **CERTTRANSBLOB** structure defines a byte buffer used to store and request certificates, transmit responses, manipulate Unicode strings, and marshal property values.

```
typedef struct _CERTTRANSBLOB {
  ULONG cb;
  [size_is (cb), unique] BYTE* pb;
} CERTTRANSBLOB;
```

cb: Unsigned integer value that MUST contain the length of the buffer pointed to by **pb**, in bytes.

pb: The **BYTE** buffer that MUST contain the binary contents being transported in this **CERTTRANSBLOB**. That content MAY consist of any of the following entities:

- A certificate.
- A certificate request.

- CA properties.
- Any common structure defined in section 2.2.1 other than VARIANT and **CERTVIEWRESTRICTION**.
- Any common structure defined in [MS-WCCE] section 2.2.1.

The **CERTTRANSBLOB** structure is empty when **cb** is set to 0 and **pb** is set to NULL.

The marshaling of other structures that can be passed in the **pb** byte buffer of **CERTTRANSBLOB** is defined in [MS-WCCE] section 2.2.1.

All instances of **CERTTRANSBLOB** used by this protocol MUST use the marshaling rules described in the following sections or in [MS-WCCE] section 2.2.1.

2.2.1.5 CATRANSPROP

The **CATRANSPROP** structure encapsulates information about a CA property. The CATRANSPROP structure and the marshaling of one or more CATRANSPROP structures into a **CERTTRANSBLOB** structure is specified in [MS-WCCE] section **2.2.2.2**.

2.2.1.6 CAINFO

Defines a basic informational block describing a CA. The structure of **CAINFO** is specified in [MS-WCCE] section **2.2.2.3**. The marshaling of CAINFO into a **CERTTRANSBLOB** structure is specified in [MS-WCCE] section 2.2.2.1.5.

2.2.1.7 CERTTRANSDBCOLUMN

The **CERTTRANSDBCOLUMN** structure is encoded within a **CERTTRANSBLOB** structure. The **CERTTRANSDBCOLUMN** structure contains **schema** information about a particular database column associated with a specific table to the client (upon the client's query via invocation of **EnumViewColumn** or **EnumViewColumnTable** methods of the **ICertAdminD** and **ICertAdminD2** interfaces, respectively).

```
typedef struct _CERTTRANSDBCOLUMN {
  DWORD Type;
  DWORD Index;
  DWORD cbMax;
  ULONG obwszName;
  ULONG obwszDisplayName;
} CERTTRANSDBCOLUMN;
```

Type: This field describes the column. The high and low WORDs are split and used separately.

Column Indexing WORD																**Reserved Byte**								**Column Value Type Byte**							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**20**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**30**	**1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I	0								See below							

Column Indexing WORD																Reserved Byte								Column Value Type Byte							
0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1

The high WORD of the **Type** field is a bit field. Only bit 15 is used. If set to 1, it indicates that the column is indexed for look-up purposes.

The low WORD of the **Type** field is divided into two bytes.

- The high byte of the low WORD MUST be set to 0 and MUST be ignored by the server upon receipt.
- The low byte of the low WORD MUST specify the value type for the column associated with a specific table using one of the following values:

Value	Meaning
0x01	The Column type is a signed integer.
0x02	The Column type is a Date.
0x03	The Column type is binary data.
0x04	The Column type is a string.

Index: Unsigned integer value that specifies the identifier for the column in the server database.

cbMax: Unsigned integer value that specifies the maximum size of data (in bytes) that this column can contain.

obwszName: Integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this column can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0.

obwszDisplayName: Integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the display name of this column can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0.

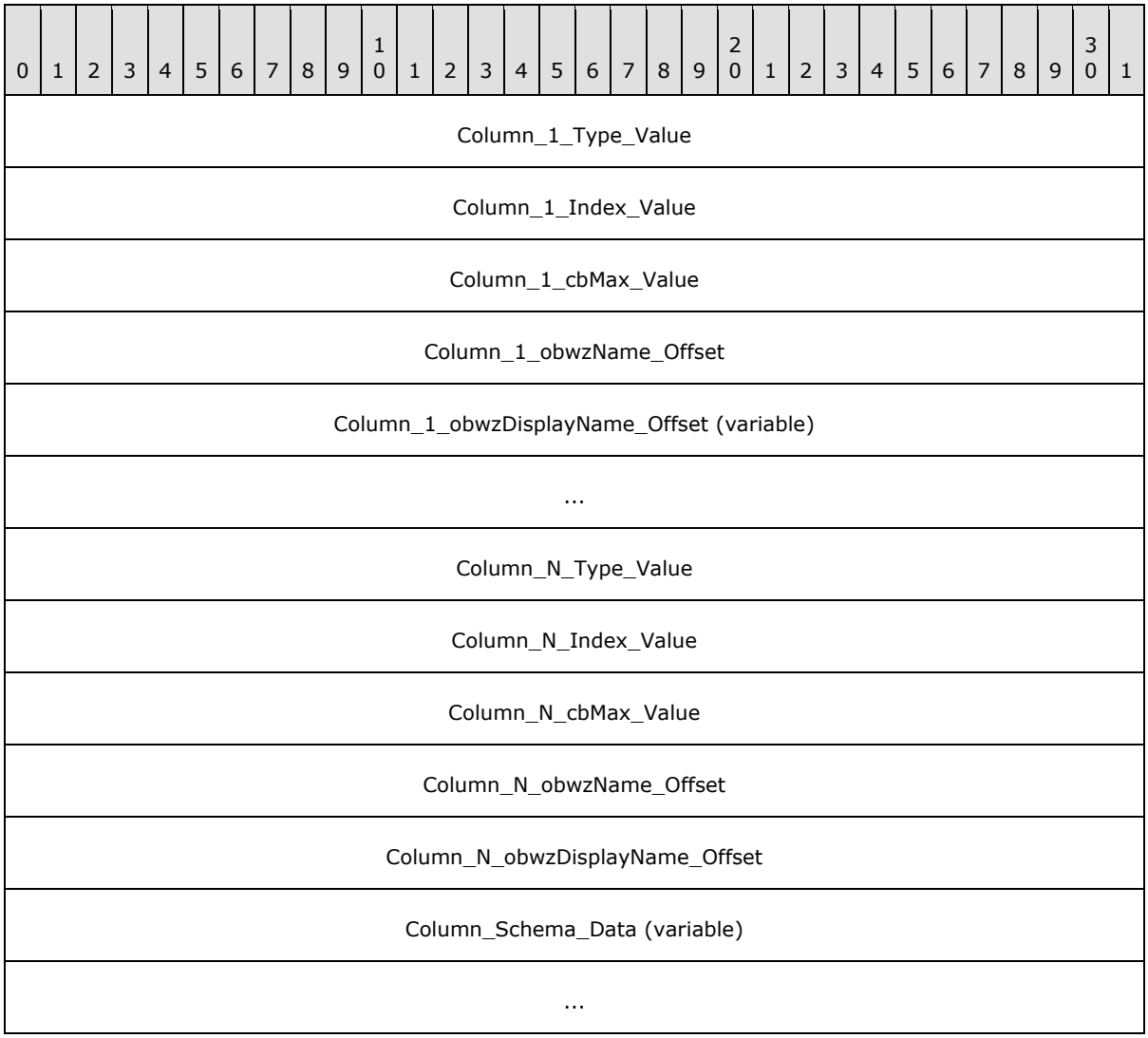
2.2.1.7.1 CERTTRANSDBCOLUMN Marshaling Format

The **CERTTRANSDBCOLUMN** structure is encoded within the byte array referenced by the **pb** member of a CERTTRANSBLOB structure.

A packet containing an array of N **CERTTRANSDBCOLUMN** structures as specified below.

The value of "N" is a separate return parameter for the EnumViewColumn and EnumViewColumnTable methods.

The **Type**, **Index**, **cbMax**, **obmszName**, and **obwszDisplayName** fields comprise the **CERTTRANSDBCOLUMN** struct. These structs MUST be contiguous and MUST NOT be padded. All structs MUST appear prior to any information on the column schema data (which appears in the Column Schema Data byte array at the end of the packet).



Column_1_Type_Value (4 bytes): The value indicating the type for the first column. The value MUST be little-endian encoded.

Column_1_Index_Value (4 bytes): The value indicating the ID for the first column. The value MUST be little-endian encoded.

Column_1_cbMax_Value (4 bytes): The maximum length of data this column can contain. The value MUST be little-endian encoded.

Column_1_obwzName_Offset (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this column can be found. The string format is NULL terminated

Unicode string. The offset mod 4 MUST be equal to 0. The offset value MUST be little-endian encoded.

Column_1_obwzDisplayName_Offset (variable): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the display name of this column can be found. The string format is NULL terminated Unicode string. The offset mod 4 MUST be equal to 0. The offset value MUST be little-endian encoded.

Column_N_Type_Value (4 bytes): The value indicating the type for the Nth column. The value MUST be little-endian encoded.

Column_N_Index_Value (4 bytes): The value indicating the ID for the Nth column. The value MUST be little-endian encoded.

Column_N_cbMax_Value (4 bytes): The maximum length of data this column can contain. The value MUST be little-endian encoded.

Column_N_obwzName_Offset (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this column can be found. The string format is NULL terminated Unicode string. The offset mod 4 MUST be equal to 0. The offset value MUST be little-endian encoded.

Column_N_obwzDisplayName_Offset (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the display name of this column can be found. The string format is NULL terminated Unicode string. The offset mod 4 MUST be equal to 0. The offset value MUST be little-endian encoded.

Column_Schema_Data (variable): Contains the schema data for all columns referenced by the obwzName and obwszDisplayName fields of the **CERTTRANSDBCOLUMN** structs. Schema data for an individual column MUST not overlap with any other data. Arbitrary padding can be inserted between data values. Schema data MUST be little-endian encoded for each character of the UNICODE null-terminated string.

2.2.1.8 CERTTRANSDBATTRIBUTE

The **CERTTRANSDBATTRIBUTE** structure is encoded within a **CERTTRANSBLOB** structure. The **CERTTRANSDBATTRIBUTE** structure is used by the server to return attribute information associated with a request to the client (upon the client's query via invocation of the **EnumAttributesOrExtensions** method of **ICertAdminD** interface).

```
typedef struct _CERTTRANSDBATTRIBUTE {
  ULONG obwszName;
  ULONG obwszValue;
} CERTTRANSDBATTRIBUTE;
```

obwszName: Integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0.

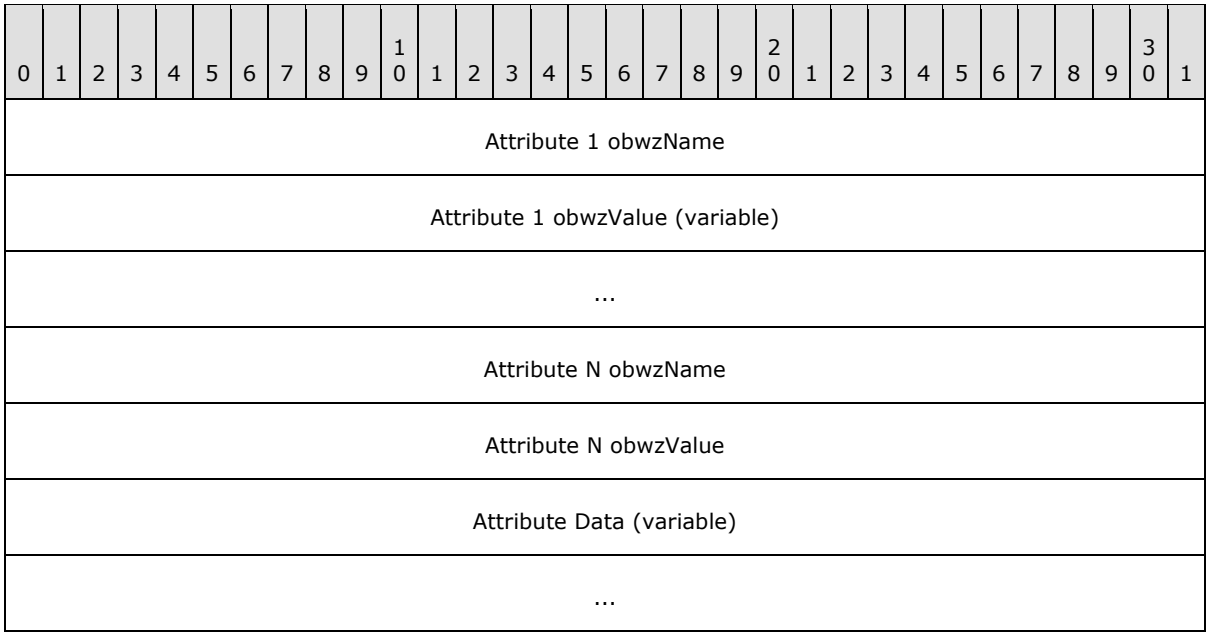
obwszValue: Integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the value of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0.

2.2.1.8.1 CERTTRANSDBATTRIBUTE Marshaling Format

A packet containing an array of N **CERTTRANSDBATTRIBUTE** structures is specified as follows.

The value of "N" is a separate return parameter for the EnumAttributesOrExtensions method.

The **obwzName** and **obwzValue** fields constitute the Attribute header. Attribute headers MUST be contiguous and MUST NOT be padded. All Attribute headers MUST appear prior to any column data (which appears in the Attribute Data byte array at the end of the packet).



Attribute 1 obwzName (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0. The offset MUST be little-endian encoded.

Attribute 1 obwzValue (variable): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the value of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0. The offset MUST be little-endian encoded.

Attribute N obwzName (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 must be equal to 0. The offset MUST be little-endian encoded.

Attribute N obwzValue (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string

containing the value of this attribute can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0. The offset MUST be little-endian encoded.

Attribute Data (variable): Contains the data for all attributes. Data for individual attributes MUST not overlap with any other attribute data. Arbitrary padding MAY be inserted between data values. Attribute name and value are string type. The data MUST use little-endian encoding format for UNICODE null-terminated string.

2.2.1.9 CERTTRANSDBEXTENSION

The **CERTTRANSDBEXTENSION** structure is encoded within a **CERTTRANSBLOB** structure. The **CERTTRANSDBEXTENSION** structure is used by the server to return certificate extension information, as specified in [RFC3280] section 4, that is associated with a request, to the client (upon the client's query via invocation of **EnumAttributesOrExtensions** method of the **ICertAdminD** interface).

```
typedef struct _CERTTRANSDBEXTENSION {
  ULONG obwszName;
  LONG ExtFlags;
  DWORD cbValue;
  ULONG obValue;
} CERTTRANSDBEXTENSION;
```

obwszName: Unsigned integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the string representation of an OID (as specified in [X680]) of this extension can be found. The string format is null-terminated UNICODE. The offset mod 4 MUST equal 0.

ExtFlags: Integer value that specifies the flags associated with the extension. The following table shows its contents.

Reserved Byte								Origin Byte								Reserved Byte								Flags							
										1										2										3	
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	**1**
0								See table below								0								0	0	0	0	0	0	D	C

The **Origin Byte** MUST equal one of the following values:

Value	Meaning
1	The extension comes from the request.
2	The extension was added by the CA.<7>
3	The extension was added by the CA.<8>
4	The extension was added by the CA.<9>

Value	Meaning
5	The extension was in the preceding certificate, as in the case of a renewal.
6	The extension comes from a PKCS7 request.
7	The extension comes from a CMC request.
8	The extension comes from the CA certificate.

The **Flags** byte uses the least 2 significant bits:

- C is the "ExtensionCriticalFlag" as defined in section 3.2.1.3.
  - A value of 0 means the extension is not critical.
  - A value of 1 means the extension is critical.
- D is the "ExtensionEnabledFlag" as defined in section 3.2.1.3.
  - A value of 0 means the extension is not disabled.
  - A value of 1 means the extension is disabled.

cbValue: Unsigned integer value containing the length of data (in bytes) referenced by *obValue* parameter.

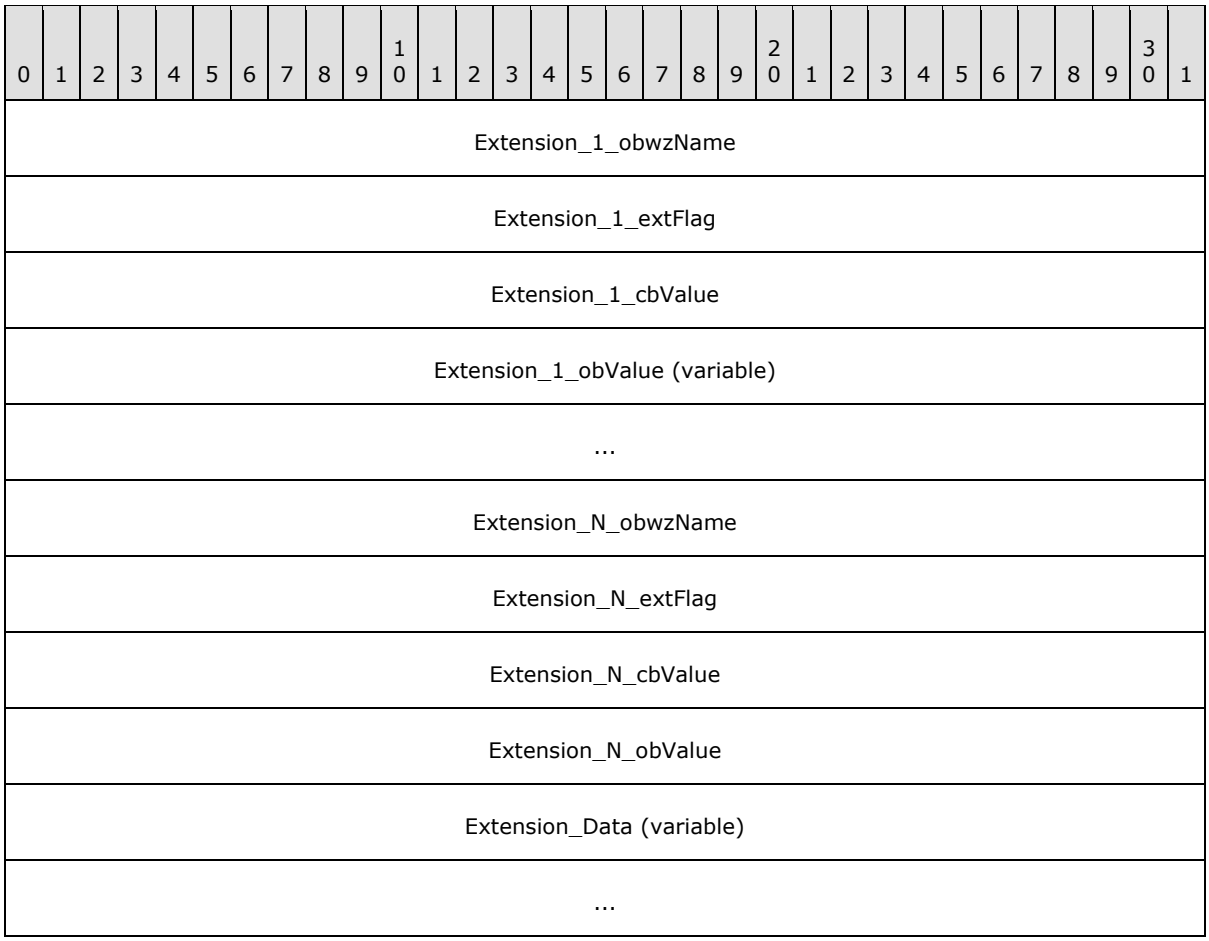
obValue: Unsigned integer containing the offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the value for this extension can be found. The length of the value is specified in **cbValue** field. The value is in ASN.1 DER (as specified in [X660]) encoded format for the extension. The offset mod 4 MUST equal 0.

2.2.1.9.1 CERTTRANSDBEXTENSION Marshaling Format

A packet containing an array of N **CERTTRANSDBEXTENSION** structures is specified below.

The value of "N" is a separate return parameter for the EnumAttributesOrExtensions method.

The **obwzName**, **extFlag**, **cbValue**, and **obValue** fields comprise the Extension header. Extension headers MUST be contiguous and MUST NOT be padded. All Extension headers MUST appear prior to any Extension data (which appears in the Extension Data byte array at the end of the packet).



Extension_1_obwzName (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this **CERTTRANSDBEXTENSION** can be found. The offset MUST be little-endian encoded. The offset mod 4 MUST equal 0.

Extension_1_extFlag (4 bytes): Integer value that specifies the flags associated with the extension. The value MUST be little-endian encoded.

Extension_1_cbValue (4 bytes): The length of the data in Extension #1 referenced by **obValue** (offset). The value MUST be little-endian encoded.

Extension_1_obValue (variable): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the value for this extension can be found. The offset mod 4 MUST equal 0. The offset value MUST be little-endian encoded.

Extension_N_obwzName (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the string containing the name of this extension can be found. The offset MUST be little-endian encoded. The offset mod 4 MUST equal 0.

Extension_N_extFlag (4 bytes): Integer value that specifies the flags associated with the extension. The value MUST be little-endian encoded.

Extension_N_cbValue (4 bytes): The length of the data in Extension N referenced by **obValue** (offset). The length value MUST use little-endian encoding format.

Extension_N_obValue (4 bytes): The offset from the beginning of the byte array buffer pointed to by the **pb** field in the containing **CERTTRANSBLOB** structure to where the value for this extension can be found. The offset mod 4 MUST equal 0. The offset value MUST be little-endian encoded.

Extension_Data (variable): Contains the data for all extensions. Data for individual extensions MUST not overlap with any other extension data. Arbitrary padding MAY be inserted between data values. An extension name data MUST use little-endian encoding format for UNICODE null-terminated string. An extension value data is a byte array.

2.2.1.10 CERTTRANSDBRESULTCOLUMN

The **CERTTRANSDBRESULTCOLUMN** structure is encoded within a **CERTTRANSBLOB** structure. The **CERTTRANSDBRESULTCOLUMN** structure is used by the server to return the result of a CA database query done by the client (upon the client's query via invocation of **OpenView** or **EnumView** methods of **ICertAdminD** interface). This structure contains data specific to a specific column in a specific row.

The **OpenView** and **EnumView** methods return data in the form of a CERTTRANSBLOB structure whose **pb** member points to an array of one or more **CERTTRANSDBRESULTROW** structures. Each **CERTTRANSDBRESULTROW** structure contains one or more **CERTTRANSDBRESULTCOLUMN** structures.

The **CERTTRANSDBRESULTCOLUMN** structure contains data for a specific column in a specific row.

```
typedef struct _CERTTRANSDBRESULTCOLUMN {
  DWORD Type;
  DWORD Index;
  ULONG obValue;
  DWORD cbValue;
} CERTTRANSDBRESULTCOLUMN;
```

Type: This field describes the column. The high and low WORDs are split and used separately.

Column Indexing WORD																**Reserved Byte**								**Column Value Type Byte**							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**20**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**30**	**1**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I	0								See below							

The high WORD of the **Type** field is a bit field. Only bit 15 is used. If set to 1, it indicates that the column is indexed for lookup purposes.

The low WORD of the **Type** field is divided into two bytes.

- The high byte of the low WORD MUST be set to 0 and MUST be ignored by the server upon receipt.
- The low byte of the low WORD MUST specify the value type for the column associated with a specific table using one of the following values:

Value	Meaning
0x01	The Column type is a signed integer.
0x02	The Column type is a Date.
0x03	The Column type is binary data.
0x04	The Column type is a string.

Index: Unsigned integer value that specifies the identifier for the column in the relevant table.

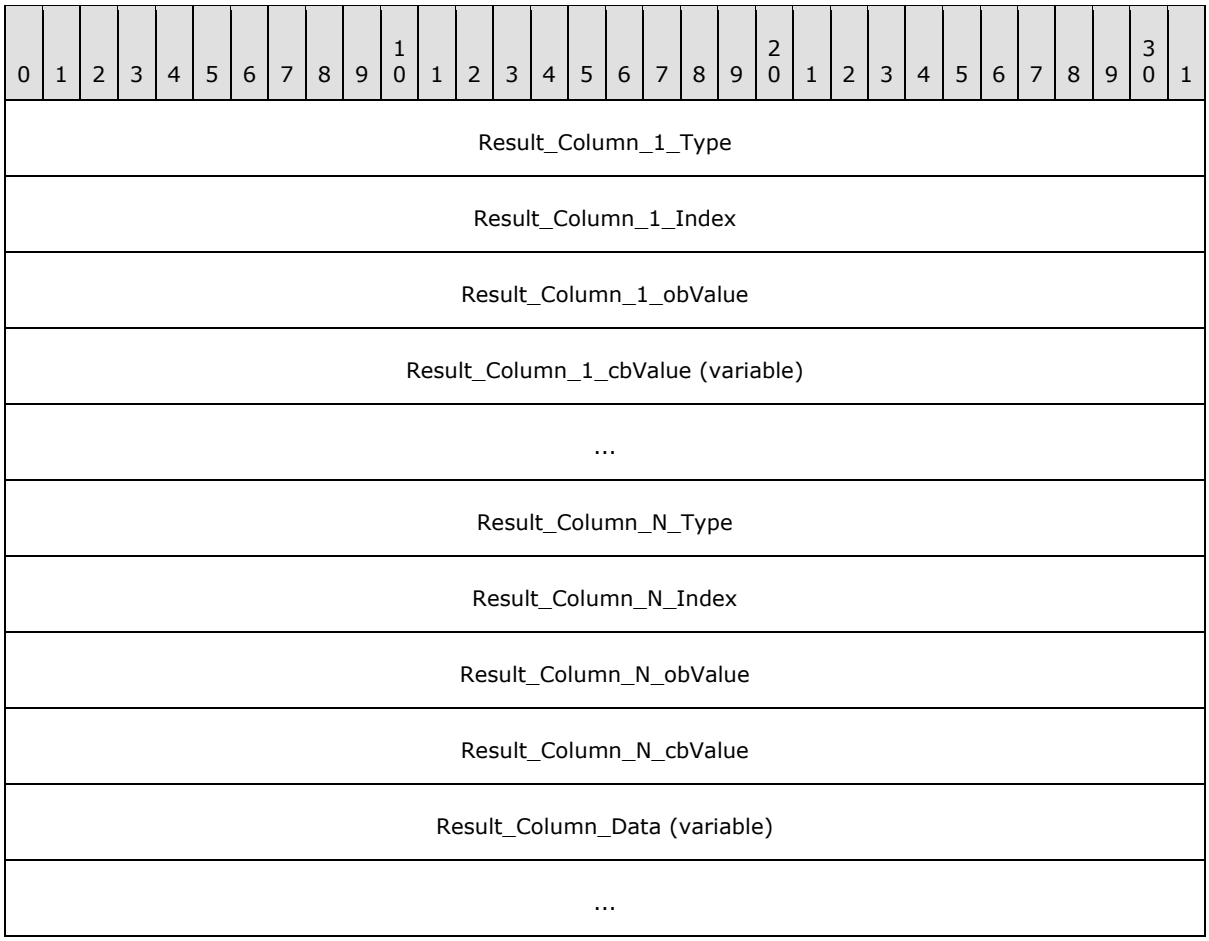
obValue: Unsigned integer containing the offset from the beginning of the corresponding **CERTTRANSDBRESULTROW** structure to where the value for this column can be found. The length of the value is specified in **cbValue** field. The offset MUST be DWORD aligned.

cbValue: Unsigned integer value that specifies the length of value (for the specific column) in bytes.

2.2.1.10.1 CERTTRANSDBRESULTCOLUMN Marshaling Format

The **CERTTRANSDBRESULTCOLUMN** structure is encoded within a CERTTRANSBLOB structure such that the **pb** member of the CERTTRANSBLOB points to the beginning of an array of one or more **CERTTRANSDBRESULTROW** structures, each of which contains one or more CERTTRANSDBRESULTCOLUMN structures.

A packet containing an array of N **CERTTRANSDBRESULTCOLUMN** structures is specified below. N is the value of the corresponding CERTTRANSDBRESULTROW's **cCol** member.



Result_Column_1_Type (4 bytes): The value indicating the type for the first column. The value MUST be little-endian encoded.

Result_Column_1_Index (4 bytes): The value indicating the ID for the first column. The value MUST be little-endian encoded.

Result_Column_1_obValue (4 bytes): The offset from the start of the corresponding **CERTTRANSDBRESULTROW** structure to the Result Column 1 data. The offset MUST be little-endian encoded. The offset mod 4 MUST equal 0.

Result_Column_1_cbValue (variable): The length of the data in Result Column 1 referenced by **obValue** (offset). The length value MUST be little-endian encoded.

Result_Column_N_Type (4 bytes): The value indicating the type for the N^{th} column. The value MUST be little-endian encoded.

Result_Column_N_Index (4 bytes): The value indicating the ID for the N^{th} column. The value MUST be little-endian encoded.

Result_Column_N_obValue (4 bytes): The offset from the start of the corresponding **CERTTRANSDBRESULTROW** structure to the data for Result Column N. The offset MUST be little-endian encoded. The offset mod 4 MUST equal 0.

Result_Column_N_cbValue (4 bytes): The length of the data in Result Column #N referenced by obValue (Offset). The length value MUST be little-endian encoded.

Result_Column_Data (variable): Contains the data for all columns. Data for individual columns MUST not overlap with any other column data. Arbitrary padding can be inserted between data values. Based on the value of the **Type** field, the data value for the column MUST be encoded as follows:

Type field value	Column Type	Data encoding
0x01	Integer	MUST use little-endian encoding format.
0x02	Date	MUST use little-endian encoding format.
0x03	Binary	
0x04	String	MUST use little-endian encoding format for each character of the UNICODE null-terminated string.

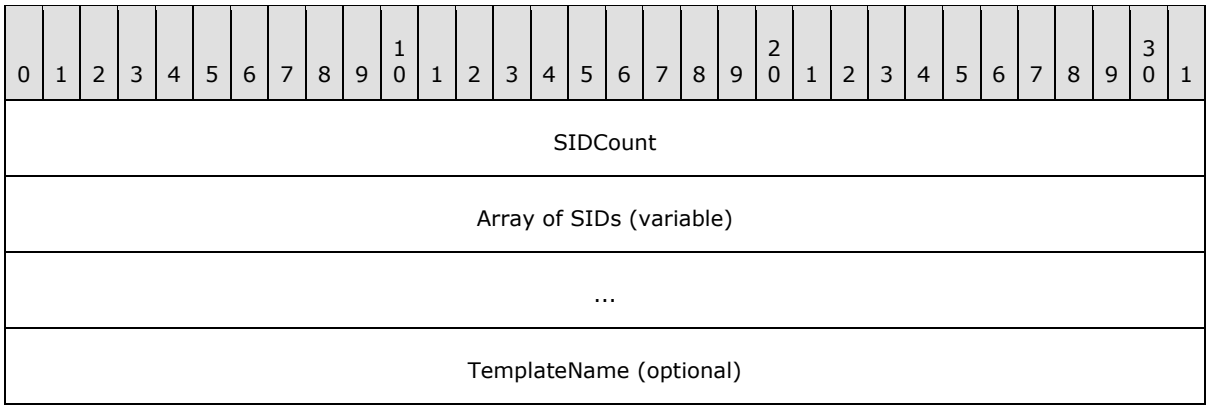
2.2.1.11 Officer and Enrollment Agent Access Rights

Officer and Enrollment Agent Access Rights structures are used by the server to return the result of a client query, for example the client's invocation of the GetOfficerRights method of the ICertAdminD2 interface.

Officer Rights and Enrollment Agent Rights are security descriptors. Security descriptor structures (**SID** structures) are defined in [MS-DTYP] section **2.4.6**. Officer Rights and Enrollment Agent Rights security descriptors have the following properties.

1. Each **access control entry (ACE)** in the **Discretionary Access Control List (DACL)** MUST have:
    1. AceType 0x9 (ACCESS_ALLOWED_CALLBACK_ACE_TYPE)
    2. AccessMask 0x0001000
2. The access control entry (ACE) contains additional application data following the **SID**.

The format for the additional application data is as follows:



SIDCount (4 bytes): A little-endian encoded **DWORD** containing the count of **SID** structures following it.

Array of SIDs (variable): An array of **SID** structures identifying either (i) **principals** for whom the officer can approve requests, or (ii) principals on whose behalf the enrollment agent can obtain certificates. For an Officer Rights security descriptor, case (i) applies. For an Enrollment Agent Rights security descriptor, case (ii) applies. **SID** structures are as defined in [MS-DTYP] section **2.4.2**.

TemplateName (4 bytes): A little-endian encoded Unicode string identifying the Common Name (CN) of the template (as defined in [MS-CRTD]) for which the officer is authorized to approve requests.

2.2.1.11.1 Marshaling Format for Officer and Enrollment Agent Rights

The marshaling of Officer rights and Enrollment Agent rights into a **CERTTRANSBLOB** structure is dependent on whether Enrollment Agent rights are supported by the server. CA implementors can determine whether to support Enrollment Agent rights. There is no requirement to do so.

If Enrollment Agent rights are not supported by the server, the **pb** member of the **CERTTRANSBLOB** structure refers to the Officer rights Security Descriptor (as defined in [MS-DTYP] section **2.4.2**), and the **cb** member contains the length of the marshaled data.

If Enrollment Agent rights are supported by the server, the **CERTTRANSBLOB** structure is created as follows:

1. If Officer rights are enabled and Enrollment Agent rights are disabled, the **pb** member of the **CERTTRANSBLOB** structure refers to following marshaled structure.

```
struct {
SECURITY_DESCRIPTOR OfficerRights;
DWORD bEARightDisabled;
};
```

Member	Value
OfficerRights	

Member	Value
bEARightsDisabled	(0x00000000)

OfficerRights: Marshaled security descriptor for Officer rights.

bEARightsDisabled: Little-endian encoded **DWORD**. Value MUST be 0x00000000.

There is no padding or **DWORD** boundary requirement.

2. If Officer rights are disabled and Enrollment Agent rights are disabled, the **pb** member of the **CERTTRANSBLOB** structure refers to following marshaled structure.

```
struct {
DWORD bEARightDisabled;
};
```

Member	Value
bEARightsDisabled	(0x00000000)

bEARightsDisabled: Little-endian encoded **DWORD**. Value MUST be 0x00000000.

There is no padding or **DWORD** boundary requirement.

3. If Officer rights are disabled and Enrollment Agent rights are enabled, the **pb** member of the **CERTTRANSBLOB** structure refers to following marshaled structure.

```
struct {
DWORD bEARightEnabled;
SECURITY_DESCRIPTOR EnrollmentAgentRights;
};
```

Member	Value
bEARightsEnabled	Must be nonzero.
EnrollAgentRights	

bEARightsEnabled: Little-endian encoded **DWORD**. Value MUST be nonzero.

Enrollment Agent Rights: Marshaled security descriptor for Enrollment Agent rights.

There is no padding or **DWORD** boundary requirement.

4. If Officer rights are enabled and Enrollment Agent rights are enabled, the **pb** member of **CERTTRANSBLOB** structure refers to following marshaled structure.

```
struct {
```

```
SECURITY_DESCRIPTOR OfficerRights;
DWORD bEARightEnabled;
SECURITY_DESCRIPTOR EnrollmentAgentRights;
};
```

Member	Value
OfficerRights	
bEARightsEnabled	Must be nonzero.
EnrollAgentRights	

OfficerRights: Marshaled security descriptor for Officer rights.

bEARightsEnabled: Little-endian encoded **DWORD**. Value MUST be nonzero.

Enrollment Agent Rights: Marshaled security descriptor for Enrollment Agent rights.

There is no padding or **DWORD** boundary requirement.

2.2.2 Certificate Requirements

2.2.2.1 CA Exchange Certificate

The Certificate Services Enrollment Protocol requires the CA to provide a CA **exchange certificate** for the purpose of client private **key archival** during the certificate enrollment process. A CA exchange certificate must be provided in the form of an X.509 digital certificate with a corresponding private key that can be used to encrypt and decrypt data.<10>

A CA Exchange Certificate contains the following X.509v1 fields:

- Version
- Serial Number
- Signature Algorithm
- Valid From
- Valid To
- Subject
- Issuer
- Public Key

A CA Exchange Certificate contains the following X.509v3 extensions, as specified in [RFC3280] section 4.2.1:

- Authority Key Identifier

- Subject Key Identifier
- Authority Information Access
- Key Usage (Key Encipherment = 0x20)
- Subject Alternative Name
- CDP (CRL Distribution Point)
- Extended Key Usage (CA Exchange OID = 1.3.6.1.4.1.311.21.5)

2.2.2.2 Key Recovery Certificate

A Key Recovery Certificate is a prerequisite for certificate enrollment that encapsulates a private key for the purposes of **key escrow** to a CA.<11> A CA MAY use one or more locally configured and specified **key recovery certificates** to encrypt the private key of a client submitted to the CA encapsulated in a certificate enrollment request.

A Key Recovery Certificate contains the following X.509v1 fields:

- Version
- Serial Number
- Signature Algorithm
- Valid From
- Valid To
- Subject
- Issuer
- Public Key

A Key Recovery Certificate contains the following X.509v3 extensions identified in section 4.2.1 of [RFC3280]:

- Authority Key Identifier
- Subject Key Identifier
- Authority Information Access
- Key Usage (Key Encipherment = 0x20)
- Subject Alternative Name
- CDP (CRL Distribution Point)
- Extended Key Usage (Key Recovery OID = 1.3.6.1.4.1.311.21.6)

2.2.3 CERTTRANSDBRESULTROW

The **CERTTRANSDBRESULTROW** structure is encoded within a **CERTTRANSBLOB** structure. The **CERTTRANSDBRESULTROW** structure is used by the server to return the result of the database

query done by the client (upon the client's query via invocation of **OpenView** or **EnumView** methods of the **ICertAdminD** interface). This structure contains data for a specific row.

```
typedef struct _CERTTRANSDBRESULTROW {
  DWORD RowId;
  DWORD ccol;
  ULONG cbrow;
} CERTTRANSDBRESULTROW;
```

RowId: Unsigned integer value that specifies the identifier for the row.

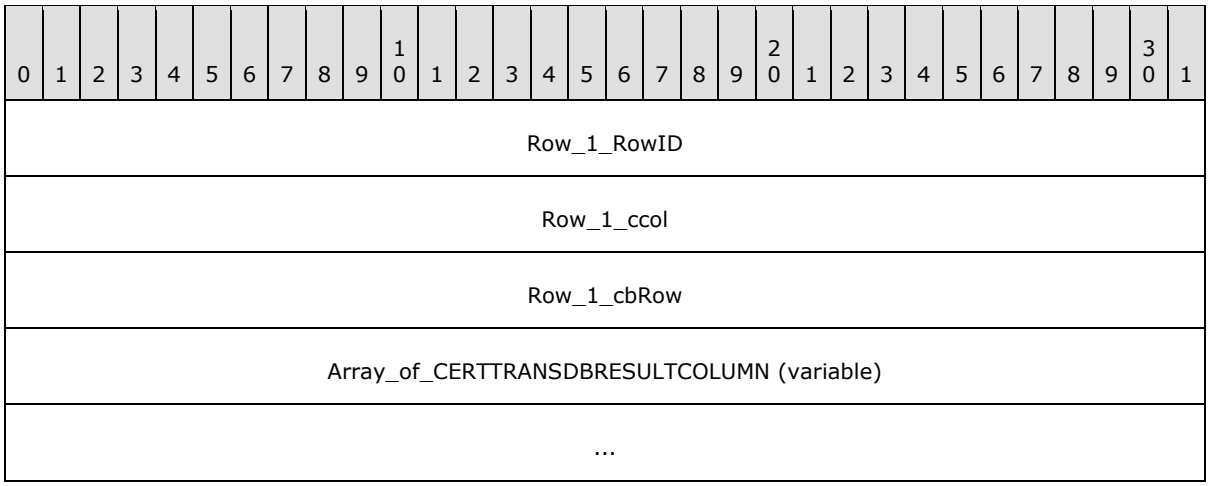
ccol: Unsigned integer value that specifies the count of **CERTTRANSDBRESULTCOLUMN** structures. Each structure contains the value of a specific column in the row identified by **RowId**.

cbrow: Unsigned integer value that specifies the total size length of row data (in bytes). This is the sum of the size of **CERTTRANSDBRESULTROW** structure, size of each **CERTTRANSDBRESULTCOLUMN** structure for the row (the count of which is specified by ccol), and the DWORD-rounded-up size of each column values.

2.2.3.1 CERTTRANSDBRESULTROW Marshaling Format

The **CERTTRANSDBRESULTROW** packet is specified below.

The **RowId**, **ccol**, **Type**, and **cbRow** fields comprise the Row Header. A complete Row header MUST appear prior to any Row data (which appears in the Row Data byte array at the end of the packet). Row data is composed of one or more **CERTTRANSDBRESULTCOLUMN** structures. The count of structures is specified in the **cCol** field value.



Row_1_RowID (4 bytes): The value identifying the row. The value MUST be little-endian encoded.

Row_1_ccol (4 bytes): The value indicating the number of columns in Row 1. The value MUST be little-endian encoded.

Row_1_cbRow (4 bytes): The total length of the Row 1 data in bytes. The length value MUST be little-endian encoded.

Array_of_CERTTRANSDBRESULTCOLUMN (variable): An array of CERTTRANSDBRESULTCOLUMN structures, as specified in section 2.2.1.10.

2.2.4 Database File Name Structure

A specific format is used for representing the names of the database files with this protocol. The string is defined as follows:

db-filename = file-type-code UNC-path

file-type-code = "!" / "D"

UNC-path = path as specified by **UNC**

The file name strings MUST be NULL-terminated Unicode strings. The file-type-code is specified as follows:

Value	Meaning	Example
'D'	Database files	"D\\server\sharepoint\...path..."
'!'	Database **log files**	"!\\server\sharepoint\...path..."

2.2.5 Common Error Codes

The following error codes are used by this protocol to indicate specific error conditions. Other error values are possible and are implementation-specific.

Value	Symbolic Name	Description
0x80070057	ERROR_INVALID_PARAMETER	The parameter is incorrect.
0x80070006	ERROR_INVALID_HANDLE	The handle is invalid.
0x8000FFFF	ERROR_UNEXPECTED_ERROR	Unexpected error occurred.
0x80004003	ERROR_INVALID_POINTER	Invalid pointer.
0x80071392	ERROR_OBJECT_EXISTS	Object already exists.
0x80094004	ERROR_REQUESTED_PROPERTYVALUE_EMPTY	An invalid parameter value was used.
0x00000001	ERROR_ARITHMETIC_OVERFLOW	Arithmetic Overflow.
0xc800020D	ERROR_INVALID_BACKUP_SEQUENCE	Invalid backup sequence.
0xc800042D	ERROR_OUT_OF_MEMORY	Out of memory.

Note Error names in this table that also appear in [MS-ERREF] have been redefined for use when implementing this protocol.

3 Protocol Details

The Certificate Services Remote Administration Protocol is a request-response protocol. The client performs a server method invocation and the server responds with the requested data or a detailed disposition value. The primary usage of this protocol is CA management. Except where specified in the following section the protocol is a single message followed by a single reply.

3.1 Client Role

3.1.1 Abstract Data Model

None.

3.1.2 Timers

The Certificate Services Remote Administration Protocol contains no timers.

3.1.3 Initialization

The Certificate Services Remote Administration Protocol depends on DCOM for authentication, as specified in [MS-DCOM].

3.1.4 Higher-Layer Triggered Events

The client administrative console<12> invokes the proper interface methods based on operator requests for the different methods available.

3.1.5 Message Processing Events and Sequencing Rules

Upon receiving a reply from the server in response to a method call, the client MUST validate the return code. Return codes from all method calls are of type **HRESULT**. If the **HRESULT** indicates success (0), the client may assume that any output parameters are present and valid. For any other return code (**HRESULT** is nonzero), the client MUST assume the method call failed. Unless redefined locally within this document, return codes are as specified in [MS-ERREF].

3.1.5.1 Processing Rules for ICertAdminD

3.1.5.1.1 ICertAdminD::SetExtension (Opnum 3)

No specific processing rules.

3.1.5.1.2 ICertAdminD::SetAttributes (Opnum 4)

No specific processing rules.

3.1.5.1.3 ICertAdminD::ResubmitRequest (Opnum 5)

No specific processing rules.

3.1.5.1.4 ICertAdminD::DenyRequest (Opnum 6)

No specific processing rules.

3.1.5.1.5 ICertAdminD::IsValidCertificate (Opnum 7)

No specific processing rules.

3.1.5.1.6 ICertAdminD::PublishCRL (Opnum 8)

No specific processing rules.

3.1.5.1.7 ICertAdminD::GetCRL (Opnum 9)

No specific processing rules.

3.1.5.1.8 ICertAdminD::RevokeCertificate (Opnum 10)

No specific processing rules.

3.1.5.1.9 ICertAdminD::EnumViewColumn (Opnum 11)

See **ICertAdminD2::EnumViewColumnTable**.

3.1.5.1.10 ICertAdminD::GetViewDefaultColumnSet (Opnum 12)

No specific processing rules.

3.1.5.1.11 ICertAdminD::EnumAttributesOrExtensions (Opnum 13)

No specific processing rules.

3.1.5.1.12 ICertAdminD::OpenView (Opnum 14)

OpenView obtains the column values for rows associated with a particular resultant set of rows from a table.

- pwszAuthority: See the pwszAuthority definition in section 3.2.5.1.1.
- acvr: Each structure in the array MUST contain the information for individual restrictions on a specific column to be applied to the associated table data before the resultant data is returned by the CA. The array of restrictions MUST be encoded as specified in **CERTVIEWRESTRICTION**.
- acolOut: Each DWORD value specifies the column identifier for the output. Column identifiers MUST be obtained by using one of the following methods: EnumViewColumn, EnumViewColumnTable, or GetViewDefaultColumnSet.

If the marshaled data buffer returned in pctbResultRows includes space for an extra **CERTTRANSDBRESULTROW** structure in addition to the row count returned in *pcelt, and if the extra **CERTTRANSDBRESULTROW** structure's **RowId** and **ccol** fields are bit-wise inverses of each other, then the field can be relied upon by the client as the total number of rows in the currently active view.

The client MUST apply the following sequencing rules:

EnumView MUST only be called after OpenView has successfully returned; otherwise, an ERROR_INVALID_HANDLE error is returned.

CloseView MUST be called to close the view that was opened via a successful call to the OpenView method; otherwise, an ERROR_INVALID_HANDLE error is returned.

EnumView MUST be called before the CloseView method call; otherwise, an ERROR_INVALID_HANDLE error is returned.

3.1.5.1.13 ICertAdminD::EnumView (Opnum 15)

See **ICertAdminD::OpenView** for client processing rules.

3.1.5.1.14 ICertAdminD::CloseView (Opnum 16)

The client SHOULD call the CloseView method to release the resources after making a successful OpenView method call.<13>

3.1.5.1.15 ICertAdminD::ServerControl (Opnum 17)

No specific processing rules.

3.1.5.1.16 ICertAdminD::Ping (Opnum 18)

The ICertAdminD::Ping method is as specified in [MS-WCCE] section 3.1.4.6.

3.1.5.1.17 ICertAdminD::GetServerState (Opnum 19)

No specific processing rules.

3.1.5.1.18 ICertAdminD::BackupPrepare (Opnum 20)

Sequencing Rules:

1. Before a **certificate services** backup can occur, a call to BackupPrepare MUST be made to notify CA that a backup of CA is about to happen.
2. The functions BackupGetAttachmentInformation and BackupGetBackupLogs MAY be used after BackupPrepare to retrieve the list of certificate services database file names and database log file names.
3. BackupOpenFile MUST be used before BackupReadFile or BackupCloseFile to open a file for backup purposes.
4. After opening the file for backup purposes (using BackupOpenFile), BackupReadFile MAY be used to retrieve the contents of the file, and call an application-specific routine to write the contents to a backup medium.
5. Before reading another file, BackupCloseFile MUST be called to close the already read file.
6. When the backup session is completed, BackupEnd MUST be invoked.

The client MUST follow the sequencing rules as described above. If BackupPrepare returns a failure, no other method calls (related to backup) MUST be made. If the sequencing rules are not met, the server returns an ERROR_UNEXPECTED_ERROR error.

The client application MUST ensure that a full backup (*grbit* parameter with value 0) has already happened before calling the server for incremental backup (*grbit* parameter with value 1).

3.1.5.1.19 ICertAdminD::BackupEnd (Opnum 21)

The client MUST enforce the sequencing rules as specified in BackupPrepare (section 3.1.5.1.18).

3.1.5.1.20 ICertAdminD::BackupGetAttachmentInformation (Opnum 22)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). After a call to BackupPrepare, the client MUST call BackupGetAttachmentInformation to obtain the list of database file names.

3.1.5.1.21 ICertAdminD::BackupGetBackupLogs (Opnum 23)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). After a call to BackupPrepare, the client MUST call BackupGetBackupLogs to obtain the list of database log file names.

3.1.5.1.22 ICertAdminD::BackupOpenFile (Opnum 24)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). The client MUST call this with the file names obtained from one of the following methods: BackupGetAttachmentInformation, BackupGetBackupLogs, and BackupGetDynamicFiles.

The client MUST remove the prefix "D" and "!" from the file names obtained via the method calls to BackupGetAttachmentInformation or BackupGetBackupLogs.

3.1.5.1.23 ICertAdminD::BackupReadFile (Opnum 25)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). After opening the file for backup purposes (by using BackupOpenFile), the client MUST call BackupReadFile to retrieve the contents of the file and call an application-specific routine to write the contents to a backup medium. The client can call this API multiple times to read the entire content of the file. Upon successful return, if the value of *pcbRead is less then cbRead or is 0, the client MUST assume that the entire contents of the file has been read and MUST call BackupCloseFile.

3.1.5.1.24 ICertAdminD::BackupCloseFile (Opnum 26)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). The client MUST call BackupCloseFile for each corresponding BackupOpenFile (and after BackupReadFile is done).

3.1.5.1.25 ICertAdminD::BackupTruncateLogs (Opnum 27)

The client MUST enforce the sequencing rules as described in BackupPrepare (section 3.1.5.1.18). The client MAY call BackupTruncateLogs<14> to truncate the log files and reduce the disk space occupied by the database files on the CA machine.

3.1.5.1.26 ICertAdminD::ImportCertificate (Opnum 28)

No specific processing rules.

3.1.5.1.27 ICertAdminD::BackupGetDynamicFiles (Opnum 29)

The client MUST retrieve a list of file names that are not part of the database but are deemed necessary as part of the backup by the CA. An example of a CA dynamic file is the certificate revocation list (CRL).

3.1.5.1.28 ICertAdminD::RestoreGetDatabaseLocations (Opnum 30)

Before performing restoration of the database, the client MUST retrieve the location where the database files have to be placed. To do so, the client MUST call RestoreGetDatabaseLocations. The location names prefixed with "D" MUST be used to restore the database files. The location names prefixed with "!" MUST be used to restore the database log files.

3.1.5.2 Processing Rules for ICertAdminD2

3.1.5.2.1 ICertAdminD2:: PublishCRLs (Opnum 31)

No specific processing rules.

3.1.5.2.2 ICertAdminD2::GetCAProperty (Opnum 32)

The ICertAdminD2::GetCAProperty is as specified in [MS-WCCE] section 3.1.4.7.

3.1.5.2.3 ICertAdminD2::SetCAProperty (Opnum 33)

No specific processing rules.

3.1.5.2.4 ICertAdminD2::GetCAPropertyInfo (Opnum 34)

The ICertAdminD2::GetCAPropertyInfo is as specified in [MS-WCCE] section 3.1.4.8.

3.1.5.2.5 ICertAdminD2::EnumViewColumnTable (Opnum 35)

No specific processing rules.

3.1.5.2.6 ICertAdminD2::GetCASecurity (Opnum 36)

No specific processing rules.

3.1.5.2.7 ICertAdminD2::SetCASecurity (Opnum 37)

No specific processing rules.

3.1.5.2.8 ICertAdminD2::Ping2 (Opnum 38)

The ICertAdminD2::Ping2 method is as specified in [MS-WCCE] section 3.1.4.6.

3.1.5.2.9 ICertAdminD2::GetArchivedKey (Opnum 39)

No specific processing rules.

3.1.5.2.10 ICertAdminD2::GetAuditFilter (Opnum 40)

No specific processing rules.

3.1.5.2.11 ICertAdminD2::SetAuditFilter (Opnum 41)

No specific processing rules.

3.1.5.2.12 ICertAdminD2::GetOfficerRights (Opnum 42)

No specific processing rules.

3.1.5.2.13 ICertAdminD2::SetOfficerRights (Opnum 43)

No specific processing rules.

3.1.5.2.14 ICertAdminD2::GetConfigEntry (Opnum 44)

No specific processing rules.

3.1.5.2.15 ICertAdminD2::SetConfigEntry (Opnum 45)

No specific processing rules.

3.1.5.2.16 ICertAdminD2::ImportKey (Opnum 46)

For the *pctbKey* parameter, the client MUST create the encrypted private key as specified in [MS-WCCE] section 3.1.4.4.4.

3.1.5.2.17 ICertAdminD2::GetMyRoles (Opnum 47)

No specific processing rules.

3.1.5.2.18 ICertAdminD2::DeleteRow (Opnum 48)

No specific processing rules.

3.1.6 Timer Events

No timers.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

This section details the data maintained by the CA.

3.2.1.1 Request Table

The Request table holds the history of all requests, both completed and pending, to the CA, one row per request.

Each column or data element in the table MUST have a database unique column **index** stored as an unsigned integer.

3.2.1.1.1 Request Table Required Data Elements

Values for the following elements are required in the request table:

- Request.Request.ID: A field used to uniquely identify the request in the table and to link to the Attribute and Extension tables.

  This field MUST have a positive value.

- Request.Raw.Request: The raw request as delivered by ICertRequestD or ICertRequestD2 (as specified in [MS-WCCE] section 2.2.2.4).

- Request.Disposition: Identifies the certificate status.

  Possible values listed below. This specification refers to these abstract values as these strings. An implementation is free to use any representation for these values.<15>

Value	Description
request failed	A certificate was never issued in response to the certificate request.
request denied	A certificate was never issued in response to the certificate request.
request pending	The request is in a state in which a certificate has not yet been issued, such as if an administrator or certificate manager is required to issue or deny a certificate based on the request.
certificate issued	A certificate was issued in response to the certificate request.
certificate revoked	A certificate was issued, and an administrator subsequently performed a revocation action on the certificate.
foreign certificate	A certificate was issued by a different CA and then imported into the CA using the **ImportCertificate** method.

- Raw.Certificate: The certificate issued for a request (if a certificate was issued).

- Request.Raw.Archived.Key: Any private key archived as part of a certificate request. Archived keys are generally encrypted with a KRA key. The format for the encrypted private key is specified in [MS-WCCE] section 3.1.1.4.2.4.

- Request.Revocation.Date: This field is used as the revocationDate for a certificate in a CRL (CRL as described in [RFC3280] section 5.1). This field is initialized as NULL and updated by the **RevokeCertificate** method.<16>

  The Request.Revocation.Date field does not follow all of the rules for **revocationDate** defined in [RFC3280] in that this field can be specified and changed at will by the CSRA client. This is achievable by client calls to the server's **RevokeCertificate** functionality described in this protocol specification. Certificates MAY be retroactively revoked; they MAY effectively be re-revoked giving any revocation date; they MAY effectively be unrevoked by re-revoking a revoked certificate while specifying a future date for the revocation, including future dates subsequent to the expiry date recorded in the certificate.

  Multiple requirements of RFC 3280 can be violated by the preceding behaviors. It is possible to have the Request.Revocation.Date differ from the "date on which the revocation occurred..." referenced in [RFC3280] section 5.1.2.6, or from "...the date at which the CA processed the revocation" referenced in [RFC3280] section 5.3.3. Changing revocation into the future beyond the date of the next CRL violates the requirements of [RFC3280] section 3.3 which states in part,

"An entry MUST NOT be removed from the CRL until it appears on one regularly scheduled CRL issued beyond the revoked certificate's validity period." Note that a CRL cannot list a certificate that is not yet revoked, as determined by its revocation date. Sufficiently retroactive revocation—that is, using a revocation date prior to the issuance of one or more existing CRLs—violates the rule that "revocation date SHOULD NOT precede the date of issue of earlier CRLs" defined in [RFC3280] section 5.3.3

- Request.Revoked.Reason: When a certificate has been revoked, this is the reason for the revocation.

  The Request.Revoked.Reason is similar to the reasonCode specified in [RFC3280] section 5.3.1, except that the protocol supports only a subset of the values the RFC defines, and it supports one additional value:

Revocation reason	Value
unspecified	0x00000000
keyCompromise	0x00000001
cACompromise	0x00000002
affiliationChanged	0x00000003
superseded	0x00000004
cessationOfOperation	0x00000005
certificateHold	0x00000006
removeFromCRL	0x00000008
Unrevoke	0xffffffff

- Serial.Number: The issued certificate serial number.
- Publish.Expired.Cert.In.CRL: This column specifies whether the certificate whose serial number is identified in Serial.Number should be included in CRLs if it is revoked, even after it has expired.

  This column is a Boolean value:

Value	Description
1	The certificate whose serial number is identified in Serial.Number should be included in CRLs if it is revoked, even after it has expired.
0	The revoked certificate should not be included in CRLs after it has expired.

3.2.1.1.2 Request Table Optional Data Elements

- Request.Key.Recovery.Hashes: Unique identifier(s) of the key recovery agent certificate(s) required to retrieve an archived private key.
- Request.Raw.Old.Certificates: In the case of a renewal, the preceding certificate.
- Request.Request.Attributes: Certificate request attributes (as defined in MS-WCCE 2.2.1.5)

- Request.Request.Type: Type or format of certificate request, such as PKCS#10 or CMS with CMC.
- Request.Request.Flags: Additional certificate request information.

  The following are example request flag values. They can be used in any combination.

Name	Value	Description
CR_FLG_FORCETELETEX	0x00000001	For encoding of the subject information in the certificate, T61String type is used for elements containing Unicode character in the value.
CR_FLG_RENEWAL	0x00000002	Certificate request is a renewal request
CR_FLG_FORCEUTF8	0x00000004	For encoding of the subject information in the certificate, UTF8String type is used for elements containing Unicode character in the value.
CR_FLG_CAXCHGCERT	0x00000008	Certificate is the CA's exchange certificate.
CR_FLG_ENROLLONBEHALFOF	0x00000010	Certificate request is an Enroll-on-behalf-of request
CR_FLG_SUBJECTUNMODIFIED	0x00000020	The subject information in the certificate is a binary copy of the subject information from the certificate request with no modification.
CR_FLG_VALIDENCRYPTEDKEYHASH	0x00000040	For certificate request with key archival, the CMC Full response will include szOID_ENCRYPTED_KEY_HASH attribute.
CR_FLG_CACROSSCERT	0x00000080	Certificate is the CA's cross certificate.
CR_FLG_ENFORCEUTF8	0x00000100	For encoding of the subject information in the certificate, UTF8String type is used for directory string elements.
CR_FLG_DEFINEDCACERT	0x00000200	Certificate request contains Authority Key Identifier extension identifying the desired CA signing key for the certificate.
CR_FLG_PUBLISHERROR	0x80000000	CA had problem publishing the certificate to the entities active directory "userCertificate" attribute.

- Request.Status.Code: Indicates whether request was successful.

  The value is 0 if the request processed successfully.

  Otherwise this field contains an error code resulting from request processing.
- Request.Disposition.Message: Text description of Request.Disposition.
- Request.Submitted.When: Time request was received by the CA.
- Request.Resolved.When: Time the CA completed request processing (whether successfully or unsuccessfully).

- Request.Revoked.When: Time the CA processed a revocation action.
- Request.Requester.Name: Requestername included in the certificate request.
- Request.Caller.Name: User or machine context that submitted the certificate request to the CA.
- Request.Signer.Policies: List of valid certificate policy OIDs for each signer certificate from the certificate request.
- Request.Signer.Application.Policies: List of valid Extended Key Usage OIDs for each signer certificate from the certificate request.
- Request.Officer: Indicates whether the caller is certificate manager of the entity corresponding to "Request.Requester.Name".
- Request.Distinguished.Name: The Distinguished name from the Subject of the certificate request.

  String representation
- Request.Raw.Name: Subject information from the certificate request.

  ASN.1 DER encoded
- Request.Country: The country attribute of the distinguished name from the Subject of the certificate request.
- Request.Organization: The organization attribute of the distinguished name from the Subject of the certificate request.
- Request.Org.Unit: The organizational-unit attribute of the distinguished name from the Subject of the certificate request.
- Request.Common.Name: The common name attribute of the distinguished name from the Subject of the certificate request.
- Request.Locality: The locality attribute of the distinguished name from the Subject of the certificate request.
- Request.State: The state or province name attribute of the distinguished name from the Subject of the certificate request.
- Request.Title: The title attribute of the distinguished name from the Subject of the certificate request.
- Request.Given.Name: The given name attribute of the distinguished name from the Subject of the certificate request.
- Request.Initials: The initials attribute of the distinguished name from the Subject of the certificate request.
- Request.SurName: The surname attribute of the distinguished name from the Subject of the certificate request.
- Request.Domain.Component: The domainComponent attribute of the distinguished name from the Subject of the certificate request.
- Request.Email: The EmailAddress attribute of the distinguished name from the Subject of the certificate request.

- Request.Street.Address: The street address attribute of the distinguished name from the Subject of the certificate request.
- Request.Unstructured.Name: The unstructured name attribute of the distinguished name from the Subject of the certificate request.
- Request.Unstructured.Address: The unstructured address attribute of the distinguished name from the Subject of the certificate request.
- Request.Device.Serial.Number: The device serial number attribute of the distinguished name from the Subject of the certificate request.
- Request.ID: RequestID corresponding to an issued certificate.
- Certificate.Hash: SHA1 hash over the value of the Raw.Certificate column.
- Certificate.Template: extnValue of extension with OID 1.3.6.1.4.1.311.20.2 of issued certificate.
- Enrollment.Flags: Values defined in "EnrollmentFlags" from MS-CRTD.
- General.Flags: Values defined in "GeneralFlags" from MS-CRTD.
- Issuer.Name.Id: Sequential number that indicates which CA key signed the issued certificate.
- Not.Before: Validity->notBefore (RFC 3280 section 4.1.2.5) field of the issued certificate.
- Not.After: Validity->notAfter (RFC 3280 section 4.1.2.5) field of the issued certificate.
- Subject.Key.Identifier: SubjectKeyIdentifier (RFC 3280 section 4.2.1.2) extension of the issued certificate.
- Raw.Public.Key: SubjectPublicKeyInfo->subjectPublicKey field of the issued certificate.
- Public.Key.Length: Length of the SubjectPublicKeyInfo->subjectPublicKey field of the issued certificate.
- Public.Key.Algorithm: SubjectPublicKeyInfo->algorithm->algorithm field of the issued certificate.
- Raw.Public.Key.Algorithm.Parameters: SubjectPublicKeyInfo->algorithm->parameters field of the issued certificate.
- UPN: UPN alternate name entry from SubjectAltName extension in the certificate.
- Distinguished.Name: Subject (RFC 3280 section 4.1.2.6) field of issued certificate.

  String representation
- Raw.Name: Subject information of the issued certificate.

  ASN.1 DER encoded
- Country: country attribute of certificate Subject.
- Organization: organization attribute of certificate Subject.
- Org.Unit: organizational-unit attribute of certificate Subject.
- Common.Name: common name attribute of certificate Subject.

- Locality: locality attribute of certificate Subject.
- State: state or province name attribute of certificate Subject.
- Title: title attribute of certificate Subject.
- Given.Name: given name attribute of certificate Subject.
- Initials: initials attribute of certificate Subject.
- SurName: surname attribute of certificate Subject.
- Domain.Component: domainComponent attribute of certificate Subject.
- Email: RFC822 Name from Subject Alternative Name of issued certificate.
- Street.Address: street address attribute of certificate Subject.
- Unstructured.Name: unstructured name attribute of certificate Subject.
- Unstructured.Address: unstructured address attribute of certificate Subject.
- Device.Serial.Number: serial number attribute of certificate Subject.

3.2.1.2 Attribute Table

A request includes an arbitrary number of attributes, as specified in [MS-WCCE] section 2.2.1.5, that are parsed and listed in the Attribute table.

Each column in the table MUST have a database unique column index stored as an unsigned integer.

Each entry in the table represents an attribute associated with a request and has the following required data elements:

- Attribute.Request.Id: This is the same unique identifier as the Request.Request.ID associated with a request in the Request table.
- Attribute.Name: Contains the name of the attribute.

  For all the rows that have the same Attribute.Request.Id, the Name MUST be unique across those rows in the table.
- Attribute.Value: Contains the value of the attribute.

3.2.1.3 Extension Table

A request includes an arbitrary number of X.509v3 extensions, as specified in [X509]. These are parsed and listed in the Extension table Each column in the table MUST have a database unique column index stored as an unsigned integer.

Each entry in the table represents an extension associated with a request and has the following required data elements:

- Extension.Request.Id: This is the same unique identifier as the Request.Request.ID associated with a request in the Request Table.
- Extension.Name: object identifier (OID), as specified in [X680], that contains the name of the extension. The ExtensionName must be the string representation of the OID associated with the

extension. Two rows with the same value in the ExtensionName field MUST NOT have the same value in the RequestId field.

- Extension.Value: Value of the extension.

  DER encoded ASN.1 (as specified in [X660] and [X690]) value of the extension.

- Extension.Raw.Value: Value of extension
- Extension.Flags: A flag that indicates the following:
  - Whether the given extension is to be marked critical
  - Whether this extension is enabled

3.2.1.4 CRL Table - Holding Certificate Revocation List Information

If a CA has a CRL table, each column in the table MUST have a database unique column index stored as an unsigned integer.

Each entry in the table represents a CRL and has the following properties associated with it, organized in columns.

3.2.1.4.1 CRL Table Required Data Elements

- CRL.Row.Id: This is the unique identifier for the CRL in the table.
- CRL.Name.Id: Sequential number that indicates which CA key the CRL is for. For example, if a CA certificate has been renewed with a new key 3 times, and the CA issues a CRL for each key, the **CRLNameId** field can be used to distinguish among the 4 issued CRLs.
- CRL.Raw.CRL: The CRL that was issued.
- CRL.Min.Base : This number identifies the CRL, complete for a given scope, that was used as the starting point in the generation of this delta CRL.

  If the value is 0, the CRL is a base CRL

  If the value is not 0, the CRL is a delta CRL
- CRL.Publish.Status.Code: This is an informational field that identifies whether the CA was able to publish the CRL to locations external to the CA Server.
- CRL.This.Publish: Time at which a CRL is first created and published.
- CRL.Propagation.Complete: Estimated time when the CRL is expected to have propagated to all servers after publishing. This data element is updated upon CRL creation and is used when creating a delta CRL. The delta CRL will be based on the last Base CRL to have completely propagated.

3.2.1.4.2 CRL Table Optional Data Elements

- CRL.Number : Sequential number that is incremented each time a new base CRL is created. (Note: if the CA creates a base CRL for multiple CA keys at once, then the CRLs for all the CA keys will have the same CRL number.)
- CRL.Count: Count of CRL entries in the CRL.

- CRL.This.Update: Value of the thisUpdate field of the CRL.
- CRL.Next.Update: Value of the nextUpdate field of the CRL.
- CRL.Next.Publish: The value of the nextPublish extension of the CRL.
- CRL.Effective: Only for delta CRLs, the time from which the CRLMinBase is valid
- CRL.Publish.Attempts: Number of times a CRL was published.
- CRL.Publish.Flags: Additional CRL information.

  Any of the following values, in any combination:

Flag	Description
CPF_BASE	Indicates a base CRL
CPF_DELTA	Indicates a delta CRL
CPF_COMPLETE	CRL published successfully
CPF_MANUAL	If caller initiated the generation of the CRL (via PublishCRL or PublishCRLs)
CPF_SHADOW	Indicates a delta CRL
CPF_BADURL_ERROR	Indicates a bad URL encountered during publishing of the CRL.
CPF_FILE_ERROR	Indicates a bad File URI encountered during publishing of the CRL.
CPF_HTTP_ERROR	Indicates a HTTP URI encountered during publishing of the CRL.
CPF_FTP_ERROR	Indicates a FTP URI encountered during publishing of the CRL.
CPF_LDAP_ERROR	Indicates a bad LDAP URI encountered during publishing of the CRL.
CPF_POSTPONED_BASE_LDAP_ERROR	Indicates delta CRL publishing was postponed during to failure in publishing of base CRL.
CPF_POSTPONED_BASE_FILE_ERROR	Indicates delta CRL publishing was postponed during to failure in publishing of base CRL.
CPF_SIGNATURE_ERROR	Indicates error in verifying the signature of the CRL generated.
CPF_CASTORE_ERROR	Indicates error in storing the generated CRL locally.

- CRL.Last.Published: Time at which CRL was last published.
- CRL.Publish.Status.Code: This is an informational field that identifies whether the CA was able to publish the CRL to locations external to the CA Server.

Contains 0 if the CRL was published successfully. Otherwise, contains the return code from a failed publish attempt. Common error codes are as specified in section 2.2.5; other error values are possible and are implementation-specific.

- CRL.Publish.Error: Contains error string associated with a non-0 CRL.Publish.Status.Code.

  Contains NULL if the CRL was published successfully.

  Otherwise, contains the return code from a failed publish attempt:

In addition to the data tables maintained on the CA, additional configuration data may be maintained in the form hierarchical view.<17> The column identifier (index) corresponding to each column in the data table MUST be unique across all the data tables.<18>

3.2.1.5 Schema Table

The Schema table contains information about the data elements defined above and their properties, and any vendor-defined elements.

The schema table has the following required elements:

- Schema.Column.Name: Internal Name of column.

  ADM element names can be used, or another unique name.

- Schema.Column.Display.Name: Display name of column.
- Schema.Column.ID: Indicates the database unique numeric identifier of the column.
- Schema.Column.Type: Indicates the data type in the column.

  This data element has four possible values:

Value	Data type
1	integer
2	date
3	binary
4	string

- Schema.Column.Type.Index: Indicates whether or not this column is indexed.

  This data element has two values:

Value	Description
0	column is not indexed
1	column is indexed

- Schema.Column.Type.Max.Value: The maximum size of data (in bytes) that the column can contain.
- Schema.Column.In.View: Specifies which view(s) (or "default column set(s)") the column is in.

This data element can have any of the following values, in any combination:

- Issued certificates
- Pending requests
- Failed requests
- Extensions
- Attributes
- CRLs
- Revoked Certificates

3.2.1.6 Datum - DB View

Every CA that conforms to this specification MUST maintain a datum called Config.Database.View.Open, as defined in [MS-WCCE] section 3.2.1.1.4,for each DCOM instance. Config.Database.View.Open indicates whether a caller has opened a view to the database. This datum has two possible values:

- **False**
- **True**

Every CA that conforms to this specification MUST initialize the value of Config.Database.View.Open to the default value of **False**.

3.2.1.7 Permissions

The CA SHOULD store the following sets of permissions, as defined in [MS-WCCE] section 3.2.1.1.4:

- Config.Permissions.CA.Security
- Config.Permissions.Officer.Rights
- Config.Permissions.Enrollment.Agent.Rights

The CA MAY enforce a caller for any of the methods specified in 3.2.5 to possess specific permissions.<19>

3.2.1.8 CRL Publishing Locations

If the CA implementer has chosen to implement revocation using CRLs, then the CA SHOULD maintain a list of one or more Config.CA.CDP.Publish.To locations, as defined in [MS-WCCE] section 3.2.1.1.4.

Each CRL Publishing Location SHOULD be a valid local file path, UNC network path, or http, ftp, ldap or file address. The CA SHOULD publish CRLs to these locations.<20>

3.2.1.9 CRL Validity Period

If the CA implements revocation via CRL, the CA SHOULD maintain a list of the following data elements to describe validity periods for CRLs. The data elements are defined in [MS-WCCE].

- Config.Base.CRL.Validity.Period
- Config.Base.CRL.Overlap.Period
- Config.Delta.CRL.Validity.Period
- Config.Delta.CRL.Overlap.Period
- Config.CA.Clock.Skew.Minutes

3.2.2 Timers

There are no timers in this protocol.

3.2.3 Initialization

Interface Initialization: The CA MUST register the DCOM interfaces and begin listening on the DCOM ports as specified in [MS-DCOM].

Cryptographic Initialization: The CA MUST have access to the **signing certificate** and exchange private keys. In addition, the CA MUST validate the CA signing certificate and its chain. The validation is based on chain validation as specified in [RFC3280].

Database Initialization: The CA MUST ensure that the database is available for use and that the tables and fields that are defined in section 3.2.1 exist.

3.2.4 Higher-Layer Triggered Events

No triggered events.

3.2.5 Message Processing Events and Sequencing Rules

The Certificate Services Remote Administration Protocol defines the following interfaces.

Interface	Description
ICertAdminD	Defines methods that enable a client to manage a certificate authority.
ICertAdminD2	Extends the **ICertAdminD** interface.

3.2.5.1 Processing Rules for ICertAdminD

The **ICertAdminD** interface provides an application programming interface for a client <21> to manage a certificate authority.

The **ICertAdminD** interface inherits the **IUnknown** interface.

The version number for **IUnknown** is 1.0. The UUID (universally unique identifier) for the ICertAdminD interface is "d99e6e71-fc88-11d0-b498-00a0c90312f3". Method opnum field values start with 3; opnum values 0 through 2 represent the IUnknown methods: QueryInterface, AddRef, and Release methods, respectively, as specified in [MS-DCOM].

Methods in RPC Opnum Order

Method	Description
SetExtension	The **SetExtension** method allows adding, modifying, or disabling extensions, as specified in [RFC3280]; a CA can include an extension in an issued certificate for a particular pending request. Opnum: 3
SetAttributes	The **SetAttributes** method sets attributes in the specified pending certificate request. Opnum: 4
ResubmitRequest	The **ResubmitRequest** method submits the specific pending certificate request to certification authority. Opnum: 5
DenyRequest	The **DenyRequest** method denies a specific certificate request that is pending. Opnum: 6
IsValidCertificate	The **IsValidCertificate** method verifies the certificate against the certification authority key and checks that the certificate has not been revoked. Opnum: 7
PublishCRL	The **PublishCRL** method sends a request to the CA Server to publish a new certificate revocation list (CRL), as specified in [RFC3280] section 5. Opnum: 8
GetCRL	The **GetCRL** method retrieves the current certificate revocation list (CRL) for the CA server. Opnum: 9
RevokeCertificate	The **RevokeCertificate** method revokes a certificate either immediately or on a specified date. Opnum: 10
EnumViewColumn	The **EnumViewColumn** method returns an array of column information. Opnum: 11
GetViewDefaultColumnSet	The **GetViewDefaultColumnSet** method returns an array of the column set identifiers associated with a specific view. Opnum: 12
EnumAttributesOrExtensions	The **EnumAttributesOrExtensions** method is used to access sets of attributes or extensions for a particular row ID. Opnum: 13
OpenView	The **OpenView** method opens a view into the database and returns a set of resultant rows data. Opnum: 14
EnumView	The **EnumView** method returns a set of resultant rows data for the opened view. Opnum: 15

Method	Description
CloseView	The **CloseView** method closes a view previously opened with the **OpenView** method call. Opnum: 16
ServerControl	The **ServerControl** method is used to force the CA server to unregister the ICertAdminD and ICertAdminD2 interfaces. Opnum: 17
Ping	The **Ping** method is used to test whether Certificate Server is alive. Opnum: 18
GetServerState	The **GetServerState** method is used to validate "Read". Opnum: 19
BackupPrepare	The **BackupPrepare** method is used to prepare the database for performing further backup operations. Opnum: 20
BackupEnd	The **BackupEnd** method completes the backup process started via a call to **ICertAdminD::BackupPrepare**. Opnum: 21
BackupGetAttachmentInformation	The **BackupGetAttachmentInformation** method is used to query the CA for the names of database files that should become part of the backup file set. Opnum: 22
BackupGetBackupLogs	The **BackupGetBackupLogs** method queries the CA for the names of database transaction log files that should become part of the backup file set. Opnum: 23
BackupOpenFile	The **BackupOpenFile** method opens a file for backup. Opnum: 24
BackupReadFile	The **BackupReadFile** method reads the database file and loads the content into the buffer provided. Opnum: 25
BackupCloseFile	The **BackupCloseFile** method closes the database file that was initialized by a prior call to **BackupOpenFile**. Opnum: 26
BackupTruncateLogs	The **BackupTruncateLogs** method function eliminates redundant records from the log files and reduces the disk storage space used by log files. Opnum: 27
ImportCertificate	The **ImportCertificate** method imports a certificate into the CA database. Opnum: 28
BackupGetDynamicFiles	The **BackupGetDynamicFiles** method retrieves the list of CA

Method	Description
	dynamic file names that need to be backed up. Opnum: 29
RestoreGetDatabaseLocations	The **RestoreGetDatabaseLocation** method retrieves the list of CA server database location names for all the database files being restored. Opnum: 30

All methods MUST NOT throw exceptions.

3.2.5.1.1 ICertAdminD::SetExtension (Opnum 3)

The **SetExtension** method allows adding, modifying, or disabling extensions, as specified in [RFC3280]. A CA can include an extension in an issued certificate for a particular pending request.

```
HRESULT SetExtension(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId,
  [in, string, unique] wchar_t const* pwszExtensionName,
  [in] DWORD dwType,
  [in] DWORD dwFlags,
  [in, ref] CERTTRANSBLOB* pctbValue
);
```

pwszAuthority: A NULL-terminated Unicode string that contains the name of the certificate server. The pwszAuthority is a Unicode string in the form of a **distinguished name** value such as "CAName" where CAName MUST be the full common name (CN) or sanitized name, as specified in [MS-WCCE] section 3.4.1, of the certificate authority (CA).

dwRequestId: A 32 bit non-zero unsigned integer value that specifies the ID of the certificate request.

pwszExtensionName: A null-terminated UNICODE string that specifies the OID for the extension to set as specified in [X680]. The string MUST be 31 or fewer non-NULL characters in length.

dwType: An unsigned integer value that specifies the type of extension being set. The *dwType* parameter MUST agree with the data type of the pb member of the *pctbValue* parameter. This parameter can be one of the following values.

Value	Meaning
0x00000001	Unsigned long data
0x00000002	Date/time
0x00000003	Binary data
0x00000004	UNICODE

dwFlags: An unsigned integer value that specifies the flags for the extension being set. This parameter can be one of the following values:

Value	Meaning
1	This is a critical extension.
2	The extension MUST NOT be used on issued certificates.

pctbValue: A pointer to a **CERTTRANSBLOB** structure. The pb member MUST point to the binary data for the extension and the cb field MUST contain the length of the value in bytes. Depending on the value of the *dwType* parameter the format of the binary data pointed by **pb** member is given in the following table.

Value of dwType	Meaning
0x00000001	The **CERTTRANSBLOB** structure **pb** member MUST point to an unsigned long data value in little-endian format.
0x00000002	The **CERTTRANSBLOB** structure **pb** member MUST point to data using little-endian encoding format.
0x00000003	The **CERTTRANSBLOB** structure **pb** member MUST point to an Array of bytes not in need of endian forcing.
0x00000004	The **CERTTRANSBLOB** structure **pb** member MUST point to a Unicode null-terminated string in little-endian format.

This method instructs the CA to add, modify or disable an extension associated with a previously submitted certificate request that is in a pending state, as specified in [MS-WCCE] section 3.2.4.1.1.1.3.

The following processing rules apply:

1. The CA MUST look up the request based on the provided *dwRequestId* parameter in the Request table. If the request is not found, the CA MUST fail the request.
2. If the request is found in the CA database the CA MUST verify that the value of the Request.Disposition column is "request pending". If the value of the Request.Disposition column is not "request pending" the CA MUST fail the request.
3. The CA MUST verify if the *pwszExtensionName* parameter contains a valid OID (as specified in [X680]). If invalid, the CA MUST fail the request.
4. The CA MUST associate the specified extension and flags with the pending request, for possible inclusion later in the issued certificate by adding the entry in the Extension table.
   - If dwType is 3 (PROPTYPE_BINARY), the CA MUST save the specified value as the extension value.
   - If dwType is 1 (PROPTYPE_LONG), the CA MUST encode the LONG value into an ASN.1 integer (as specified in [X660]) and save the encoded value as the extension value.
   - If dwType is 2 (PROPTYPE_DATE), the CA MUST encode the FILETIME value into an ASN.1 Choice-of-Time (as specified in [X660]) and save the encoded value as the extension value.
   - If dwType is 4 (PROPTYPE_STRING), the CA MUST encode the UNICODE string value into an IA5String (as specified in [X660]) and save the encoded value as the extension value.

- If dwType is any other value, the CA MUST fail the request. The error code SHOULD be ERROR_INVALID_PARAMETER (0x80070057).

3.2.5.1.2 ICertAdminD::SetAttributes (Opnum 4)

The **SetAttributes** method sets attributes in the specified pending certificate request.

```
HRESULT SetAttributes(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId,
  [in, string, unique] wchar_t const* pwszAttributes
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

dwRequestId: A 32 bit non-zero unsigned integer value that specifies the ID of the certificate request.

pwszAttributes: A null-terminated UNICODE string. The value of the string MUST have same format as specified in [MS-WCCE] section 3.2.4.1.1.1.

This method instructs the CA to add or modify a name value pair associated with a previously submitted certificate request that is in a pending state. Information about a pending certificate request is specified in [MS-WCCE] section 3.2.4.1.1.1.4.2.

The following processing rules apply:

1. The CA MUST look up the request based on the provided *dwRequestId* parameter in the Request table. If the request is not found, the CA MUST fail the request.
2. If the request is found in the CA database the CA MUST verify that the value of the Request.Disposition column is "request pending". If the value of the Request.Disposition column is not "request pending" the CA MUST fail the request.
3. The CA MUST parse the **pwszAttributes** string as done for **ICertRequestD::Request** and **ICertRequestD2::Request2** methods, as specified in [MS-WCCE] section 3.2.4.1.2.
4. The CA MUST ignore invalid name-value pair entries.
5. The CA MUST associate the valid name-value pair entries with the pending request, for possible impact later on the issued certificate by adding the entries in the Attribute table.

3.2.5.1.3 ICertAdminD::ResubmitRequest (Opnum 5)

The **ResubmitRequest** method submits the specific pending certificate request to CA.

```
HRESULT ResubmitRequest(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId,
  [out] DWORD* pdwDisposition
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

dwRequestId: A 32 bit non-zero unsigned integer value that specifies the ID of the certificate request.

pdwDisposition: A pointer to an unsigned integer value that receives the disposition status of the certificate (upon resubmission).

This method instructs the CA to try again to process a previously submitted certificate request, which is in a pending or denied state.

The following processing rules apply:

1. The CA MUST validate that the Unicode string referenced by *pwszAuthority* matches (case-insensitive) the full common name (defined in [MS-GLOS]) or the sanitized name of the CA. Sanitized Name is defined in [MS-WCCE] sections 1.3.2.3 and 3.4.1. If the value fails to match, the server MUST fail the request. The error code SHOULD be 0x80070057.
2. The CA MUST look up the request based on the provided *dwRequestId* parameter in the Request table.
   - If the request is not found, the CA MUST place 0x80094004 in the *pdwDisposition* parameter and return successfully.<22>
   - If the request is found in the CA database, the row will be referenced as the "identified row" in the following processing rules.
   - The CA MUST verify that the value of the Request.Disposition column in the identified row is "request pending" or "request denied".
   - If the value of the Request.Disposition column in the identified row is not "request pending" or "request denied", the CA MUST place 0x80094003 in the *pdwDisposition* parameter and return successfully.
   - If the value of the Request.Disposition column in the identified row is "request denied" and the invoker of the method is not the CA administrator, the CA MUST place 0x80094003 in the *pdwDisposition* parameter and return successfully.
3. The CA MUST try to process the request as if it is a new request. Information on request processing is as specified in [MS-WCCE] section 3.2.4.1.1.1.4.
4. If the request processing results in the CA issuing the certificate, the CA MUST set the Request.Disposition column of in the identified row to certificate issued, and place 3 in the *pdwDisposition* parameter and return successfully.
5. If the request processing results in the CA denying the certificate, the CA MUST set the Request.Disposition column of in the identified row to request denied. The CA SHOULD place a non-zero value in the *pdwDisposition* parameter and return successfully.
6. If the request processing results in the CA pending the certificate, the CA MUST return 5 in the *pdwDisposition* parameter and return successfully.

3.2.5.1.4 ICertAdminD::DenyRequest (Opnum 6)

The **DenyRequest** method denies a specific certificate request that is pending.

```
HRESULT DenyRequest(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

dwRequestId: A 32 bit non-zero unsigned integer value that specifies the ID of the certificate request.

The following processing rules apply:

1. The CA MUST look up the request based on the provided *dwRequestId* parameter in the Request table. If the request is not found, the CA MUST fail the request. If the request is found, the row selected will be referred to as the identified row in the following processing rules.
2. If the value of the Request.Disposition column in the identified row is not "request pending", the CA MUST fail the request.
3. Otherwise, the CA MUST set the value of the Request.Disposition column in the identified row to "request denied".

3.2.5.1.5 ICertAdminD::IsValidCertificate (Opnum 7)

The **IsValidCertificate** method verifies the certificate against the CA key and checks that the certificate has not been revoked.

```
HRESULT IsValidCertificate(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, string, unique] wchar_t const* pSerialNumber,
  [out] LONG* pRevocationReason,
  [out] LONG* pDisposition
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

pSerialNumber: A null-terminated UNICODE string specifying a serial number that identifies the certificate to be reviewed. The string MUST specify the serial number as an even number of hexadecimal digits. If necessary, a zero can be prefixed to the number to produce an even number of digits. The string MUST NOT contain more than one leading zero. Information on the serial number is as specified in [RFC3280] section 4.1.2.2.

pRevocationReason: A pointer to a **LONG** value that receives the revocation reason code. The revocation reason code MUST be one of the following values defined for CRLReason, as specified in [RFC3280] section 5.3.1

Value	Meaning
0	unspecified
1	keyCompromise

Value	Meaning
2	cACompromise
3	affiliationChanged
4	superseded
5	cessationOfOperation
6	certificateHold

pDisposition: A pointer to a **LONG** that receives the disposition status of the request. This parameter can be one of the following values:

Value	Meaning
0x00000002	The certificate has been revoked.
0x00000003	The certificate is still valid.
0x00000004	The certificate was never issued.

The following processing rules apply:

1. Unless otherwise specified below, the value returned as pRevocationReason SHOULD be 0.
2. The CA MUST look up a row in the RequestTable where the value of the Serial.Number column is identical to the value provided in the *pSerialNumber* parameter.
   - If a row was not found, the CA MUST return 4 in the *pDisposition* parameter.
   - If a row was found, this row will be referred to as the "identified row" in the following processing rules.
3. The CA MUST inspect the value of the Request.Disposition column in the identified row and apply the following rules:
   - If the value is "certificate issued", the CA MUST return 3 in the *pDisposition* parameter.
   - If the value is "certificate revoked" and the value in Request.Revocation.Date is greater than the current time, the CA MUST return 3 in the *pDisposition* parameter
   - If the value is "certificate revoked" and the value in Request.Revocation.Date contains a value that is less than or equal to the current time, the CA MUST return 2 in the pDisposition parameter, read the Request.Revoked.Reason property from the Request table, and return the value as the pRevocationReason argument.

3.2.5.1.6 ICertAdminD::PublishCRL (Opnum 8)

The **PublishCRL** method sends a request to the CA Server to publish a new certificate revocation list (CRL) if the CA has chosen to implement revocation via CRL.

```
HRESULT PublishCRL(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] FILETIME FileTime
```

```
);
```

pwszAuthority: See the pwszAuthority definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

FileTime: Contains a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). Specifies the nextUpdate value of the CRL as specified in [RFC3280], section 5 in Coordinated Universal Time (Greenwich Mean Time).

The following processing rules apply:

1. If the CA has not implemented revocation via CRL, the CA MUST return an error when this method is invoked.
2. If the CA has implemented revocation via CRL, when this method is invoked, the CA MUST create a new base CRL for each valid CA certificate.
   - If the CA has enabled delta CRLs, as indicated by a nonzero Config.Delta.CRL.Validity.Period value, the CA SHOULD create a new delta CRL in addition to a new base CRL.
3. The CA server SHOULD check the Request table to determine which certificates to include in a CRL.
   - If Request.Revocation.Date is NOT NULL and is in the past:
     1. If the Request.Disposition is not “certificate revoked”, do not include the certificate Serial.Number on the CRL.

        One implication of this rule is that, if a previously revoked certificate is **unrevoked** (such as when it is removed from certificate hold) and expires between the times two successive regularly published CRLs are published, then the certificate’s serial number will not appear on the latter CRL. This outcome violates [RFC3280] section 3.3, "An entry MUST NOT be removed from the CRL until it appears on one regularly scheduled CRL issued beyond the revoked certificate's validity period."

     2. If the Request.Disposition is “certificate revoked”:
        1. For a delta CRL, exclude certificates whose Request.Revoked.When is before the CRL.This.Update of the oldest unexpired base CRL.
        2. If the Publish.Expired.Cert.In.CRL column is set to 1, the certificate Serial.Number SHOULD be included in CRLs regardless of the certificate’s notAfter time.
        3. If the Publish.Expired.Cert.in.CRL column is not set to 1:
           1. If the certificate’s notAfter time is before the CRL.This.Publish of the last CRL, do not include the certificate’s Serial.Number on the CRL.
           2. If the certificate’s notAfter time is after the CRL.This.Publish of the last CRL, include the certificate’s Serial.Number on the CRL.
4. At CRL creation time, the CA SHOULD create a new CRL table entry, populating the following data elements:
   - CRL.Number

- CRL.Name.Id
- CRL.Min.Base (only for a delta CRL)
- CRL.Count
- CRL.This.Publish
- CRL.This.Update : The CA SHOULD use the current time minus Config.CA.Clock.Skew.Minutes<23>. The value of CRL.This.Update MUST NOT be less than the NotBefore date of the current CA certificate. If the calculated value of CRL.This.Update is less than the NotBefore date of the current CA certificate is, the CA SHOULD replace the CRL.This.Update value with the NotBefore date of the current CA certificate.
- CRL.Next.Publish: The CA SHOULD set this data element to the current time plus Config.Base.CRL.Validity.Period for a base CRL or Config.Delta.CRL.Validity.Period, for a delta CRL.
- CRL.Effective (for delta CRL)
- CRL.Propagation.Complete: The CA SHOULD set this data element to the current time plus overlap period. <24>
- If a FileTime is provided, which is indicated by a nonzero FileTime value, the CA MUST use it in following manner:
  - If the FileTime value is less than the current time, the CA MUST fail with an error code ERROR_INVALID_PARAMETER.
  - Otherwise, the CA MUST use the value supplied in the FileTime parameter as the CRL.Next.Update.
- If a FileTime is not provided, the CA SHOULD set CRL.Next.Update to the current time plus Config.Base.CRL.Validity.Period for a base CRL or Config.Delta.CRL.Validity.Period, for a delta CRL, plus overlap period<25>. The CA MAY add a clock skew. <26>

5. Once the new CRL table entry is created, the CA SHOULD create and publish a CRL file for each valid CA key to such locations as have been identified in Config.CA.CDP.Publish.To. <27>The CA SHOULD allocate data to the various CRL extensions as follows:
   - CRL.This.Update SHOULD be used for the thisUpdate field of the CRL.
   - CRL.Next.Update SHOULD be used for the nextUpdate field of the CRL.
   - For each certificate determined in rule 3 above to be included on the CRL, Serial.Number and Request.Revocation.Date SHOULD be used respectively as the userCertificate and revocationDate components of the revoked certificates list in the CRL.
   - CRL.Next.Publish SHOULD be used for the nextPublish extension of the CRL.
   - CRL.Number SHOULD be used for the CRL Number extension of CRL.
   - CRL.Min.Base SHOULD be used for the Delta CRL indicator extension of CRL (for a delta CRL only).
   - Otherwise, CRL fields and extensions are as specified in [RFC3280] section 5.
6. After publishing, the CA SHOULD update the following CRL table data elements:

- CRL.Publish.Status.Code
- CRL.Publish.Error
- CRL.LastPublished (for a base CRL only): The CA SHOULD set this data element to the current time minus the lifetime of the current base CRL.
- CRL.Publish.Attempts: The CA SHOULD set this data element to one to indicate the publish attempt for the newly created CRL.

3.2.5.1.7 ICertAdminD::GetCRL (Opnum 9)

The **GetCRL** method instructs the CA to return the current base CRL for the current CA key.

```
HRESULT GetCRL(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out, ref] CERTTRANSBLOB* pctbCRL
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

pctbCRL: If the function succeeds, this method MUST return a **CERTTRANSBLOB** structure that contains the ASN.1 DER (as specified in [X660] and [X690]) encoded CRL (CRLRawCRL) for the CA server's current signing certificate.

The **GetCRL** method instructs the CA to return the recent Base CRL (CRLRawCRL), which is signed with the current CA key to the caller.

3.2.5.1.8 ICertAdminD::RevokeCertificate (Opnum 10)

The **RevokeCertificate** method revokes a certificate either immediately or on a specified date. It instructs the CA to revoke a certificate based on the certificate's serial number and reason code.

```
HRESULT RevokeCertificate(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, string, unique] wchar_t const* pwszSerialNumber,
  [in] DWORD Reason,
  [in] FILETIME FileTime
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

pwszSerialNumber: A null-terminated UNICODE string specifying a serial number that identifies the certificate to be revoked. The string MUST specify the serial number as plain hexadecimal digits (no leading 0x) as specified in [RFC3280] section 4.1.2.2.<28>

Reason: An unsigned integer value that specifies the revocation reason code. The revocation reason code MUST be either one of the values listed below (and specified in [RFC3280] section 5.3.1), or one of the following values: 0xfffffffd, 0xfffffffe, or 0xffffffff.

Value	Meaning
0	unspecified
1	keyCompromise
2	cACompromise
3	affiliationChanged
4	superseded
5	cessationOfOperation
6	certificateHold
8	removeFromCRL
0xfffffffd	See processing rules, beginning with rule 2.
0xfffffffe	See processing rules, beginning with rule 3.
0xffffffff	See processing rules, beginning with rule 4.

FileTime: Contains a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (Coordinated Universal Time (UTC)). This value specifies the date, according to Greenwich Mean Time, on which the certificate became invalid. The *FileTime* corresponds to the **Revocation_Date** defined in section 3.2.1.1.

The following processing rules apply:

1. The CA MUST find a row in the Request database that contains the certificate that needs to be revoked in this method invocation. It finds it by comparing the value of the *pwszSerialNumber* parameter to the values of the Serial.Number column in the Request table. This is a case-sensitive string comparison. If none of the rows in the Request table has a Serial.Number value identical to the value passed in the *pwszSerialNumber* parameter, the CA MUST fail the request. The error returned SHOULD be 0x80070057. After a row with an identical serial number, as specified above, is found, the CA SHOULD continue with the processing rules below.

   Note The processing rules below refer to this row as the "identified row."

2. If the *Reason* parameter is 0xfffffffd, the CA MUST set the Publish.Expired.Cert.In.CRL column of the identified row to 0 and return successfully.<29>

3. If the *Reason* parameter is 0xfffffffe, the CA MUST set the Publish.Expired.Cert.In.CRL column of the identified row to 1 and return successfully.<30>

4. If the value for the *Reason* parameter is 0xffffffff and the value of the Request.Revoked.Reason column in the identified row is not 6, the CA MUST fail the request, and the error code SHOULD be ERROR_INVALID_DATA.

5. If the *Reason* parameter is not 0, 1, 2, 3, 4, 5, 6, 8, 0xfffffffd, 0xfffffffe, or 0xffffffff, the CA MUST fail the request. The error code SHOULD be E_INVALIDARG (0x80070057), as specified in [MS-ERREF] section 2.1. Otherwise, the CA MUST continue with the following processing rules.<31>

6. If the Request.Disposition column of the identified row equals "certificate issued", the CA MUST set the Request.Disposition column of the identified row to "certificate revoked".

7. If the Request.Disposition column of the identified row equals "certificate revoked":

    1. If the Request.Revoked.Reason column of the identified row equals "certificateHold":

        - If the *Reason* parameter is 0xffffffff, the CA MUST set the Request.Disposition column to "certificate issued".

        - Otherwise, the CA MUST continue with processing rules 9 and 10 below.

    2. If the Request.Revoked.Reason column of the identified row does not equal "certificateHold":

        - If the *Reason* parameter is 0xffffffff, the CA MUST fail the request, as specified in rule 4 above.

        - If the *Reason* parameter is "certificateHold", the CA MUST fail the request. The error code SHOULD be ERROR_INVALID_DATA (0x0x8007000D), as specified in [MS-ERREF] section 2.1.

        - If the *Reason* parameter does not equal 0xffffffff and does not equal "certificateHold", the CA MUST continue with processing rules 9 and 10 below.

          One implication of this rule is that the Windows CA allows updating of the Reason_Code and Revocation_Date values of a certificate that has already been revoked. While [RFC3280] section 5.3.3 describes revocationDate as the "date at which the CA processed the revocation", the Windows CA thus allows a different date to be used.

8. If the Request.Disposition column of the identified row does not equal "certificate issued" and the Request.Disposition column of the identified row does not equal "certificate revoked", the CA must fail the request. The error code SHOULD be ERROR_INVALID_DATA, as specified in [MS-ERREF], section 2.1.

9. The CA MUST set the Request.Revoked.Reason column of the identified row to the value passed in the *Reason* parameter.

10. The CA MUST set the Request.Revocation.Date column of the identified row to the value passed in the *FileTime* parameter.

3.2.5.1.9 ICertAdminD::EnumViewColumn (Opnum 11)

The **EnumViewColumn** method returns an array of column information.

```
HRESULT EnumViewColumn(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD iColumn,
  [in] DWORD cColumn,
  [out] DWORD* pcColumn,
  [out, ref] CERTTRANSBLOB* pctbColumnInfo
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

iColumn: An unsigned integer that specifies the index of the column to return. Valid values are from 0 to one less than the maximum number of columns for the Request table.

cColumn: An unsigned integer that specifies the requested count of columns to return.

pcColumn: A pointer to an unsigned integer that receives the returned count of **CERTTRANSDBCOLUMN** structures returned by the server in a *pctbColumnInfo* parameter.

pctbColumnInfo: A pointer to a **CERTTRANSBLOB** structure. Upon return, pb points to an array of the marshaled **CERTTRANSDBCOLUMN** structures as described in section **CERTTRANSDBCOLUMN**.

The **EnumViewColumn** method returns information about columns associated with the Request table to the client. The processing rules for this function are the same as for **EnumViewColumnTable** with the *iTable* parameter value set to 0x00000000.

3.2.5.1.10 ICertAdminD::GetViewDefaultColumnSet (Opnum 12)

The **GetViewDefaultColumnSet** method returns an array of the column identifiers associated with a specific view.

```
HRESULT GetViewDefaultColumnSet(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD iColumnSetDefault,
  [out] DWORD* pcColumn,
  [out, ref] CERTTRANSBLOB* pctbColumnInfo
);
```

pwszAuthority: See the *pwszAuthority* definition in **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

iColumnSetDefault: An unsigned integer value that specifies the requested default column set to get. The value MUST be one of the values in the following table. If a value other than one of those listed is used, the error E_INVALIDARG is returned.

Value	Meaning
0xFFFFFFFF	Pending requests
0xFFFFFFFE	Issued and revoked certificates in addition to failed requests
0xFFFFFFFD	Failed requests
0xFFFFFFFC	Extensions
0xFFFFFFFB	Attributes
0xFFFFFFFA	CRLs
0xFFFFFFF9	Revoked certificates

pcColumn: A pointer to the unsigned integer that receives the count of Column identifiers returned by the server in the *pctbColumnInfo* parameter.

pctbColumnInfo: A pointer to a **CERTTRANSBLOB** structure. Its **cb member** field MUST contain the length of the array (in bytes) referenced by the **pb member** field. The **pb member** field MUST point to an array of DWORDs where each DWORD value represents the identifier for a column. Each **DWORD** in the array is marshaled using little-endian format.

The CA server MUST return the array of associated column identifiers in the following manner:

Value of iColumnSetDefault parameter	Processing rule
0xFFFFFFFF	The CA MUST return a subset of column identifiers for a pending request view from the Request table.<32>
0xFFFFFFFE	The CA MUST return a subset of column identifiers for an issued certificate view from the Request table.<33>
0xFFFFFFFD	The CA MUST return a subset of column identifiers for a failed request view from the Request table.<34>
0xFFFFFFFC	The CA MUST return a subset of column identifiers for an extension view from the Extension table.<35>
0xFFFFFFFB	The CA MUST return a subset of column identifiers for an attribute view from the Attribute table.<36>
0xFFFFFFFA	The CA MUST return a subset of column identifiers for a list of CRL view from the CRL table.<37>
0xFFFFFFF9	The CA MUST return a subset of column identifiers for a revoked certificate view from the Request table.<38>

3.2.5.1.11 ICertAdminD::EnumAttributesOrExtensions (Opnum 13)

The **EnumAttributesOrExtensions** method is used to access sets of attributes or extensions for a particular row ID.

```
HRESULT EnumAttributesOrExtensions(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD RowId,
  [in] DWORD Flags,
  [in, string, unique] wchar_t const* pwszLast,
  [in] DWORD celt,
  [out] DWORD* pceltFetched,
  [out, ref] CERTTRANSBLOB* pctbOut
);
```

pwszAuthority: See the pwszAuthority definition in section **ICertAdminD::SetExtension (section 3.2.5.1.1)**.

RowId: An unsigned integer value that specifies the RequestId of the row to retrieve attributes or extensions.

Flags: An unsigned integer value that MUST take either of the following values:

Value	Meaning
0x00000000	Enumerate Attributes
0x00000001	Enumerate Extensions

pwszLast: A pointer to a NULL-terminated UNICODE string that specifies the name of the attribute or extension beyond which the data is requested. If the value of **Flags** is 1, the name MUST be an Object Identifier string as specified in [X680].

celt: An unsigned integer value that specifies the requested count of attributes (**CERTTRANSDBATTRIBUTE**) or extensions (**CERTTRANSDBEXTENSION**) structures to be returned to the client.

pceltFetched: A pointer to the unsigned integer that receives the actual count of the attributes (**CERTTRANSDBATTRIBUTE**) or extensions (**CERTTRANSDBEXTENSION**) structure data returned by the server in the *pctbOut* parameter.

pctbOut: A pointer to the **CERTTRANSBLOB** structure. The data returned is marshaled **CERTTRANSDBATTRIBUTE** or **CERTTRANSDBEXTENSION** structure array as described in **CERTTRANSDBATTRIBUTE** and **CERTTRANSDBEXTENSION**.

EnumAttributesOrExtensions obtains information about the attributes or extensions (as specified in [MS-WCCE] section 2.2.1.5) associated with a specific request in Request table.

1. The CA Server MUST apply the following ordered processing rules. If an error is encountered, the CA SHOULD return the error specified and terminate the processing of the method.

2. The CA Server MUST enforce that the *Flags* parameter is either 0 or 1; otherwise, it MUST return an error. The error SHOULD be ERROR_INVALID_PARAMETER.

3. The CA Server MUST enforce that the *RowId* parameter value is greater than 0 and that a row exists in the Request table with the specified *RowId* in the Request.Request.ID column. Otherwise, the CA Server MUST return an error code. If the *RowId* parameter value is not greater than 0, the error code SHOULD be ERROR_INVALID_PARAMETER. If the row does not exist, the error SHOULD be 0x80094004.

4. If the value of the *Flags* parameter is 0x00000000, the CA MUST compute the resultant set of rows from the Attribute table, where each row MUST have the same value in its Request_ID_Attribute column as the *RowId* parameter value. If the *pwszLast* parameter is non-NULL, then the CA MUST find the row in the resultant set with the same value in the AttributeName as the value of the *pwszLast* parameter (case-insensitive comparison). If the row is not found, the CA MUST fail. The error code SHOULD be 0x80094004. If the row is found, the CA MUST remove the rows prior to this row (including this row) from the resultant set of rows to return to the client.

   Note The computed set of rows from this step will be referred to as the resultant set in the following processing rules.

5. If the value of the *Flags* parameter is 0x00000001, the CA MUST compute the resultant set of rows from the Extension table, where each row MUST have the same value in its Request_ID_Extension column as the *RowId* parameter value. If the *pwszLast* parameter is non-NULL, the CA MUST find the row in the resultant set with the same value in the ExtensionName as the value of the *pwszLast* parameter (case-insensitive comparison). If the row is not found, the CA MUST fail. The error code SHOULD be ERROR_INVALID_PARAMETER. If the row is found, the CA MUST remove the rows prior to this row (including this row) from the resultant set of rows to return to the client.

   Note The computed set of rows from this step will be referred to as the resultant set in the processing rules below.

6. The value of the *celt* parameter is referred to as RequestedRows. If RequestedRows is smaller than the number of the rows in the resultant set, the CA MUST return in step 7 below only the first RequestedRows rows in the sorted resultant set, and remove the rest of the rows from the resultant set.

7. The value of the **pceltFetched* parameter MUST be set to the number of the rows in the resultant rows set returned in step 8below.

8. The CA MUST return the resultant set computed in step 4 or 5 above in the *pctbOut* parameter. The format and marshaling for the value of *pctbOut* is described in **CERTTRANSDBATTRIBUTE** and **CERTTRANSDBEXTENSION**.

3.2.5.1.12 ICertAdminD::OpenView (Opnum 14)

The **OpenView** method opens a view into the database and returns a set of resultant rows data.

```
HRESULT OpenView(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD ccvr,
  [in, size_is(ccvr)] CERTVIEWRESTRICTION const* acvr,
  [in] DWORD ccolOut,
  [in, size_is(ccolOut)] DWORD const* acolOut,
  [in] DWORD ielt,
  [in] DWORD celt,
  [out] DWORD* pceltFetched,
  [out, ref] CERTTRANSBLOB* pctbResultRows
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

ccvr: An unsigned integer value that specifies the count of a **CERTVIEWRESTRICTION** structure array pointed to by *acvr* parameter.

acvr: A pointer to an array of **CERTVIEWRESTRICTION** structures. For more information, see section 2.2.1.3.

ccolOut: An unsigned integer value that specifies the count of a DWORD array pointed to by the *acolOut* parameter.

acolOut: A pointer to an array of DWORDs. Each DWORD value specifies the column identifier for the resultant set of rows.

ielt: An unsigned integer value that specifies the index of the first row to return from the resultant set of rows.

celt: An unsigned integer value that specifies the requested count of the row data to be returned from the resultant set of rows.

pceltFetched: A pointer to an unsigned integer value that receives the actual count of row data returned by the server in the *pctbResultRows* parameter.

pctbResultRows: A pointer to a **CERTTRANSBLOB** structure. The *pb* byte array of the CERTTRANSBLOB structure MUST contain (on successful return) an array of one or more marshaled **CERTTRANSDBRESULTROW** (section 2.2.3) structures, each of which in turn contain one or more **CERTTRANSDBRESULTCOLUMN** (section 2.2.1.10) structures. In addition, an extra CERTTRANSDBRESULTROW structure MUST be included in the array when the server encounters the end of the enumeration as described below.

The **OpenView** method opens a view into the database and returns a set of resultant rows data.

The CA Server MUST enforce the following sequencing rules:

1. If the **OpenView** method is called when the value of Config.Database.View.Open is **False**, the server MUST set Config.Database.View.Open to **True**, and continue with the processing rules below.

2. If the **OpenView** method is called when the value of Config.Database.View.Open is **True**, the server MUST return an error. The error code SHOULD be ERROR_UNEXPECTED_ERROR.

The CA Server MUST apply the following processing rules:

1. The CA Server MUST ensure that sortOrder is specified only in one of the restrictions specified in the *acvr* parameter. If more than one column specifies sort order, the server MUST reject the request with the error ERROR_INVALID_PARAMETER.

2. The CA Server MUST also ensure that all of the column identifiers specified in the restrictions and the *acolOut* parameter are valid and associated with only one database table. The table MUST be either the Request table or the CRL table. If the table is not the Request Table or the CRL table, the CA MUST return ERROR_INVALID_PARAMETER.

3. The CA Server MUST compute the resultant set of rows to return, after applying the restrictions on the row set associated with the table and sorting the resultant rows based on the restriction information. If no sort ordering is provided in the restriction set, the sorting MUST be done based on the primary index column of the table. If no restriction set is given, then the resultant set of rows to return is the entire row set associated with the table.

   Each restriction MUST be processed in the following manner:

   1. Based on the value type of the column identified via *columnIndex* parameter, the pbValue MUST be decoded. The value MUST be in the format and encoded as specified in section 2.2.1.3; otherwise, the CA Server MUST fail with the error ERROR_INVALID_PARAMETER.

   2. Based on the seek operator specified in the restriction, the value for the associated column in each row of the resultant set MUST satisfy rules described below.

   3. For each row in the resultant set (after sorting), only the columns identified by column identifiers in acolOut array MUST be retained. The rest of the columns MUST be removed from the resultant set.

   4. See section 3.2.5.1.13 for the processing of the *ielt* and *celt* parameters, and the data returned via the *pceltFetched* and *pctbResultRows* out parameters.

   5. The value for the associated column in each row of the resultant set MUST be compared, based on the seek operator, to the value specified in the restriction. If the comparison fails, the row MUST be removed from the resultant set. For column type 3 (BINARY) seek operator MUST be 0x00000001; otherwise, the call MUST fail with error ERROR_INVALID_PARAMETER.

3.2.5.1.13 ICertAdminD::EnumView (Opnum 15)

The **EnumView** method returns a set of resultant row data for the opened view.

```
HRESULT EnumView(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD ielt,
  [in] DWORD celt,
  [out] DWORD* pceltFetched,
  [out, ref] CERTTRANSBLOB*& pctbResultRows
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

ielt: An unsigned integer value that specifies the index of the first row to return from the resultant set of rows.

celt: An unsigned integer value that specifies requested count of the row data to be returned from the resultant set of rows.

pceltFetched: A pointer to an unsigned integer value that receives the actual count of row data returned by the server in the *pctbResultRows* parameter.

pctbResultRows: A pointer to a **CERTTRANSBLOB** structure. The *pb* byte array of the CERTTRANSBLOB structure MUST contain (on successful return) an array of one or more marshaled **CERTTRANSDBRESULTROW** (section 2.2.3) structures, each of which in turn contain one or more **CERTTRANSDBRESULTCOLUMN** (section 2.2.1.10) structures . In addition, an extra CERTTRANSDBRESULTROW structure MUST be included in the array when the server encounters the end of the enumeration as described below.

The CA Server MUST enforce the following sequencing rules:

1. If the **EnumView** method is called when the value of Config.Database.View.Open is **False**, the server MUST return an error. The error code SHOULD be ERROR_INVALID_HANDLE.

2. If the **EnumView** method is called when the value of Config.Database.View.Open is **True**, the server MUST continue with the processing rules below.

The CA Server MUST apply following processing rules:

1. The CA Server MUST use the resultant set of rows as obtained via **OpenView** method call.

2. The CA Server MUST use the value of ielt as an index to this resultant set of rows.

3. The number of resultant rows returned MUST be a minimum of the *celt* parameter value and remaining number of rows in the set (starting from index). Value of **pceltFetched* parameter MUST be set to number of resultant rows returned.

4. The CA server MAY return the total row count as extra data in the returned marshaled data buffer as follows:

5. The server MUST add an extra **CERTTRANSDBRESULTROW** structure at the end of the other marshaled **CERTTRANSDBRESULTROW** rows' data. The count returned in **pctbResultRows* MUST not include the extra **CERTTRANSDBRESULTROW** structure. The extra **CERTTRANSDBRESULTROW** structure's **RowId** field MUST<39> be set to the total row count, and the **ccol** field MUST be set to its bit-wise inverse.

6. Format for placing the retrieved value(s) of database row(s) and column(s) into the CERTTRANSDBRESULTROW and CERTRANSDBRESULTCOLUMN structures referenced by *pctbResultRows* is described above. Marshaling rules for a CERTTRANSDBRESULTROW into a CERTTRANSBLOB are described in section 2.2.3.1 above. Marshaling rules for a CERTTRANSDBRESULTCOLUMN into a CERTTRANSBLOB are described in section 2.2.1.10.1 above.

3.2.5.1.14 ICertAdminD::CloseView (Opnum 16)

The **CloseView** method closes a view previously opened with the **OpenView** method call.

```
HRESULT CloseView(
```

```
  [in, string, unique] wchar_t const* pwszAuthority
);
```

pwszAuthority: See pwszAuthority definition in section 3.2.5.1.1.

The CA Server MUST release the resources associated with storing the result set of rows obtained via **OpenView** method call. <40>

The CA Server MUST enforce the following sequencing rules:

1. If the **CloseView** method is called when the value of Config.Database.View.Open is **False**, the server MUST return an error. The error code SHOULD be ERROR_INVALID_HANDLE.

2. If the **CloseView** method is called when the value of Config.Database.View.Open is **True**, the server MUST set the value of Config.Database.View.Open to **False**.

3.2.5.1.15 ICertAdminD::ServerControl (Opnum 17)

The **ServerControl** method is used to force the CA server to unregister the **ICertAdminD** and **ICertAdminD2** interfaces.

```
HRESULT ServerControl(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwControlFlags,
  [out, ref] CERTTRANSBLOB* pctbOut
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

dwControlFlags: An unsigned integer value that specifies the control to be sent to the certificate server. It MUST take following value:

Value	Meaning
0x000000001	Request unregister for DCOM interfaces for the certificate server.
0x000000002	Not currenlty used.
0x000000003	Not currenlty used.

pctbOut: Not used.

The following processing rule applies.The CA MUST check the control flags, and MUST take the following action:

1. If the control flags value is 1, the CA MUST unregister the **ICertAdminD** and **ICertAdminD2** interfaces.

2. If the control flags value is 2 or 3, the CA returns successfully.

3. If the control flags value is not 1, 2, or 3, the CA MUST return an error. The error code SHOULD be "0x80070057".

3.2.5.1.16 ICertAdminD::Ping (Opnum 18)

The **Ping** method is used to test whether the Certificate Server is alive.<41>

```
HRESULT Ping(
  [in, string, unique] wchar_t const* pwszAuthority
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

The ICertAdminD::**Ping** method is as specified in [MS-WCCE] section 3.2.4.1.1.3.

3.2.5.1.17 ICertAdminD::GetServerState (Opnum 19)

The **GetServerState** method is used to validate that the caller has permission to read the CA database.

```
HRESULT GetServerState(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out] DWORD* pdwState
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

pdwState: A pointer to an unsigned integer value that specifies whether the caller has permission to read from the CA database.

The following processing rules apply:

- The CA MUST return 1 for pdwState if the caller has permission to read the CA database. Otherwise, the CA MUST return 0.

3.2.5.1.18 ICertAdminD::BackupPrepare (Opnum 20)

The **BackupPrepare** method is used to prepare the database for performing further backup operations such as **BackupEnd**, **BackupGetAttachmentInformation**, **BackupGetBackupLogs**, **BackupOpenFile**, **BackupReadFile**, **BackupCloseFile**, and **BackupTruncateLogs**.

```
HRESULT BackupPrepare(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] unsigned long grbitJet,
  [in] unsigned long dwBackupFlags,
  [in] const WCHAR* pwszBackupAnnotation,
  [in] DWORD dwClientIdentifier
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

grbitJet: An unsigned long value. This MUST be from following values:

Value	Meaning
0x00000000	Prepare for Full backup of the CA database.
0x00000001	Prepare an incremental backup as opposed to a full backup. This means that only the log files since the last full or incremental backup will be backed up.

dwBackupFlags: An unsigned long value. Not used. MUST be 0. MUST be ignored on receipt.

pwszBackupAnnotation: Not Used. MUST be Empty string (L"'). MUST be ignored on receipt.

dwClientIdentifier: An unsigned long value. Not used. MUST be 0. MUST be ignored on receipt.

The CA Server MUST enforce the following sequencing rules.<42>

1. Before a CA backup can occur, **BackupPrepare** MUST be called by the client to notify the CA that a backup of the CA is about to happen.
    1. If **BackupGetAttachmentInformation** is called without a previous call to **BackupPrepare** with the *grbitJet* parameter set to 0 (for a full backup), the CA MUST fail. The error code SHOULD be 0xc8000230.
    2. If **BackupPrepare** is not called, then the rest of database backup related API's **BackupGetAttachmentInformation**, **BackupGetBackupLogs**, **BackupOpenFile**/**BackupReadFile**, **BackupCloseFile**, and **BackupEnd** MUST fail.
2. If **BackupPrepare** is called again by the same client (before calling **BackupEnd**), the CA MUST fail.
3. If **BackupReadFile** is called before **BackupOpenFile**, the CA MUST fail.
4. If **BackupCloseFile** is called before **BackupOpenFile**, the CA MUST fail.
5. If **BackupOpenFile** is called again (before calling **BackupCloseFile**), the CA MUST fail.
6. **BackupReadFile** MUST be called after **BackupOpenFile** and before **BackupCloseFile**; otherwise, the CA MUST fail.
7. **BackupEnd** MUST be the final API for a backup session.

If the above sequencing rules are not met, the server MUST return ERROR_UNEXPECTED_ERROR error. The CA Server MUST apply the following processing rule:

- The CA server MUST take into account the grbitJet value to account for an incremental backup versus a full backup. If a full backup has not taken place, the CA MUST return failure if the method is invoked for an incremental backup (grbitJet parameter value 1).

3.2.5.1.19 ICertAdminD::BackupEnd (Opnum 21)

The **BackupEnd** method completes the backup process started via a call to **ICertAdminD::BackupPrepare**.

```
HRESULT BackupEnd();
```

This method has no parameters.

The CA server MUST enforce the sequencing rules for **BackupEnd** as specified in section 3.2.5.1.18.<43>

3.2.5.1.20 ICertAdminD::BackupGetAttachmentInformation (Opnum 22)

The **BackupGetAttachmentInformation** method is used to query the CA for the names of database files that should become part of the backup file set.

```
HRESULT BackupGetAttachmentInformation(
  [out, size_is(, *pcwcDBFiles)]
    WCHAR** ppwszzDBFiles,
  [out] LONG* pcwcDBFiles
);
```

ppwszzDBFiles: A pointer to a WCHAR pointer that will receive the list of null-terminated database file names. Detailed database file name structure formatting information is specified in section 2.2.4.

pcwcDBFiles: A pointer to an integer value that contains the total length (in characters) of all strings (including NULL-terminator character) returned in *ppwszzDBFiles*.

The CA server MUST enforce the sequencing rules for **BackupGetAttachmentInformation** as specified in section 3.2.5.1.18.<44>

The CA Server MUST apply the following processing rule:

- The CA server MUST return a list of file names associated with the CA database data files required for backup. The files corresponding to the names that are returned MUST be accessible to the CA. Each file name MUST be in UNC format and MUST be prefixed with "D" character. If there are no database files, the CA MUST set the value of pcwcDBFiles parameter to 0 and return successfully.

3.2.5.1.21 ICertAdminD::BackupGetBackupLogs (Opnum 23)

The **BackupGetBackupLogs** method queries the CA for the names of database transaction log files that should become part of the backup file set.

```
HRESULT BackupGetBackupLogs(
  [out, size_is(, *pcwcLogFiles)]
    WCHAR** ppwszzLogFiles,
  [out] LONG* pcwcLogFiles
);
```

ppwszzLogFiles: A pointer to WCHAR pointer that will receive the list of null-terminated log file names. Detailed database file name structure formatting information is specified in section 2.2.4.

pcwcLogFiles: A pointer to an integer value that contains the total length (in characters) of all strings (including NULL terminator character) returned in *ppwszzLogFiles*.

The CA server MUST enforce the sequencing rules for **BackupGetBackupLogs**, as specified in section 3.2.5.1.18.<45>

The CA server MUST apply the following processing rule:

- The CA server MUST return a list of file names associated with the databases log files required for backup. The list of files MUST be accessible to the CA. Each file name MUST be in UNC format and MUST be prefixed with '!' character. If there are no database log files, the CA MUST set 0 as the value of *pcwcLogFiles parameter and return successfully.

3.2.5.1.22 ICertAdminD::BackupOpenFile (Opnum 24)

The **BackupOpenFile** method opens a file for backup.

```
HRESULT BackupOpenFile(
  [in, string, unique] wchar_t const* pwszPath,
  [out] unsigned hyper* pliLength
);
```

pwszPath: Null-terminated UNICODE string that specifies the path to the targeted file. The file name MUST be UNC form, for example: "\\server\sharepoint\...path...\filename.ext".

pliLength: A pointer to signed 64-bit integer that receives the size of the targeted file (in bytes).

The CA server MUST enforce the sequencing rules for **BackupOpenFile** as specified in section 3.2.5.1.18.<46>

The CA Server MUST apply the following processing rules:

1. The CA server MUST enforce that the FileName is one of the file names (without the prefix) that could be returned via call to **BackupGetAttachmentInformation** or **BackupGetBackupLogs**.
2. The CA server MUST also enforce that the file corresponding to file name MUST be accessible to the CA.
3. Upon successful return, the CA MUST return the size of the file content (in bytes) in **pliLength* parameter.

3.2.5.1.23 ICertAdminD::BackupReadFile (Opnum 25)

The **BackupReadFile** method reads the database file and loads the contents into the buffer provided. The file MUST be initialized by a prior call to **BackupOpenFile**.

```
HRESULT BackupReadFile(
  [ref, out, size_is(cbBuffer)] BYTE* pbBuffer,
  [in] LONG cbBuffer,
  [out] LONG* pcbRead
);
```

pbBuffer: A pointer to the buffer that receives the read data.

cbBuffer: The size of the above buffer (in bytes). This parameter must be a multiple of the Page Size of the operating system.

pcbRead: A pointer to an integer that receives the actual number of bytes read.

The CA server MUST enforce the sequencing rules for **BackupReadFile** as specified in section 3.2.5.1.18.<47>

The CA Server MUST apply following processing rules:

1. If **BackupReadFile** is called for the first time after **BackupOpenFile**, the CA MUST read the content (in bytes) from the start of the file up to (at maximum) the value of *cbBuffer* parameter.
2. Upon subsequent call to **BackupReadFile**, the CA MUST read the content starting from one byte offset of last byte read in the previous call to **BackupReadFile**.
3. If the CA has already reached end-of-file, the call MUST succeed with 0 as the value of *pcbRead; otherwise, *pcbRead MUST contain the actual number of bytes read from the file.

3.2.5.1.24 ICertAdminD::BackupCloseFile (Opnum 26)

The **BackupCloseFile** method closes the database file that was initialized by a prior call to **BackupOpenFile**.

```
HRESULT BackupCloseFile();
```

This method has no parameters.

The CA server MUST enforce the sequencing rules for **BackupCloseFile** as specified in section 3.2.5.1.18.

3.2.5.1.25 ICertAdminD::BackupTruncateLogs (Opnum 27)

The **BackupTruncateLogs** method function eliminates redundant records from the log files and reduces the disk storage space used by log files.

```
HRESULT BackupTruncateLogs();
```

This method has no parameters.

The CA server MAY<48> remove the redundant records in the log files (records that are present in the database also are defined as redundant), thereby decreasing the disk space used to store the log files.<49>

3.2.5.1.26 ICertAdminD::ImportCertificate (Opnum 28)

The **ImportCertificate** method imports a certificate into the CA database.

```
HRESULT ImportCertificate(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, ref] CERTTRANSBLOB* pctbCertificate,
  [in] LONG dwFlags,
  [out] LONG* pdwRequestId
);
```

pwszAuthority: See pwszAuthority definition in section 3.2.5.1.1.

pctbCertificate: A **CERTTRANSBLOB** containing a ASN.1 DER (as specified in [X660] and [X690]) encoded certificate that will be inserted into the CA database.

dwFlags: A **LONG** value. It MUST take one of the following values:

Value	Meaning
0	If this value is set, the CA server does not allow certificates that are not issued by it to be imported into its database.
FLAG_ALLOW_IMPORT_FOREIGN 0x00010000	Request CA server to allow certificates that are not issued by it, to be imported into its database.

pdwRequestId: Returns the request ID for the imported certificate; this is used to refer to the certificate after it is imported into the database.

ImportCertificate imports a certificate into the CA database RequestTable.

The CA server MUST apply the following processing rules:

1. The CA server MUST enforce that the *pctbCertificate* parameter value represents an ASN.1 DER (as specified in [X660]) encoded certificate. If not, it MUST fail with error ERROR_INVALID_DATA[0x8007000D].

2. The CA server MUST validate the signature on the certificate with the **public key** associated with the CA's signing certificates.

3. If the signature validation succeeds (at step 2) and the certificate does not already exist in the Request Table (this is checked by searching on the Serial.Number in the certificate in the database), then the certificate MUST be added to the RequestTable as a new row and the CA MUST return the resulting Request.Request.ID to the client. For processing rules for each data element in the RequestTable, see the table ImportCertificate Data Element Processing Rules below.

4. If the signature validation succeeds (at step 2) and the certificate is already present in the RequestTable, the CA MUST fail with error ERROR_OBJECT_EXISTS.

5. If the signature validation fails (at step 2) and FLAG_ALLOW_IMPORT_FOREIGN is not passed as value of *dwFlags*, the CA MUST fail with error 0x800b0107, according to the ImportCertificate Data Element Processing Rules table.

6. If the signature validation fails (at step 2) and FLAG_ALLOW_IMPORT_FOREIGN is passed as value of *dwFlags* and the certificate is already present in the RequestTable, the CA SHOULD return the resulting Request.Request.ID to the client. For processing rules for each data element in the Request table, see the table ImportCertificate Data Element Processing Rules below.<50>

7. If the signature validation fails (at step 2) and FLAG_ALLOW_IMPORT_FOREIGN is passed as value of *dwFlags* and the certificate is not already present in the RequestTable, then the certificate SHOULD be added to the RequestTable as a new row and the CA SHOULD return the resulting Request.Request.ID to the client. For processing rules for each data element in the Request table, see the table ImportCertificate Data Element Processing Rules below.<51>

Table: ImportCertificate Data Element Processing Rules

The certificate fields and extensions SHOULD be processed and stored in individual Request table fields according to the following rules:

Data type	Maximum size of data	Data element name	Processing Rule or x.509 Certificate Field or Extension
0x10001 Long indexed	4 bytes	Request.RequestID	Next sequential number after Request.Request.ID of last database row.
0x1 long	4 bytes	"Request.StatusCode"	If the import is successful, the CA SHOULD set this value to 0.
0x10001 long indexed	4 bytes	"Request.Disposition"	If the certificate being imported was issued by a foreign CA, set Request.Disposition to "foreign certificate". <52> Otherwise, set to "certificate issued".
0x4 string	4 bytes	"Request.DispositionMessage"	"certificate issued".
0x10002 date indexed	8 bytes	"Request.SubmittedWhen"	Time when the method is invoked.
0x10002 date indexed	8 bytes	"Request.ResolvedWhen"	Time when the method is invoked.
0x10004 string indexed	2048 bytes	"Request.RequesterName"	Identity of the caller invoking the method.
0x10004 string indexed	2048 bytes	"Request.CallerName"	Identity of the caller invoking the method.
0x3 bin	4096 bytes	"Request.RawName"	Raw subject information from the certificate.
0x4 string	8192 bytes	"Request.Country"	Country attribute of certificate Subject.
0x4 string	8192 bytes	"Request.Organization"	Organization attribute of certificate Subject.
0x4 string	8192 bytes	"Request.OrgUnit"	Organizational-unit attribute of certificate Subject.
0x4 string	8192 bytes	"Request.CommonName"	Common name attribute of certificate Subject.
0x4 string	8192 bytes	"Request.Locality"	Locality attribute of certificate Subject.
0x4 string	8192 bytes	"Request.State"	State or province name attribute of certificate Subject.
0x4 string	8192 bytes	"Request.Title"	Title attribute of certificate Subject.

Data type	Maximum size of data	Data element name	Processing Rule or x.509 Certificate Field or Extension
0x4 string	8192 bytes	"Request.GivenName"	Given name attribute of certificate Subject.
0x4 string	8192 bytes	"Request.Initials"	Initials attribute of certificate Subject.
0x4 string	8192 bytes	"Request.SurName"	Surname attribute of certificate Subject.
0x4 string	8192 bytes	"Request.DomainComponent"	domainComponent attribute of certificate Subject.
0x4 string	8192 bytes	"Request.EMail"	RFC 822 Name from Subject Alternative Name.
0x4 string	8192 bytes	"Request.DeviceSerialNumber"	serial number attribute of certificate Subject.
0x10001 long indexed	4 bytes	"RequestID"	Next sequential number after Request.Request.ID of last database row.
0x3 bin	16384 bytes	"RawCertificate"	Byte stream pointed to by in parameter pctbCertificate ->pb member.
0x10004 string indexed	128 bytes	"CertificateHash"	SHA1 hash over the value of "RawCertificate".
0x10004 string indexed	254 bytes	"CertificateTemplate"	extnValue of extension with OID 1.3.6.1.4.1.311.20.2.
0x10004 string indexed	128 bytes	"SerialNumber"	Serial number (RFC 3280 section 4.1.2.2) of certificate.
0x2 date	8 bytes	"NotBefore"	Validity::notBefore (RFC 3280 section 4.1.2.5) field of certificate.
0x10002 date indexed	8 bytes	"NotAfter"	Validity::notAfter (RFC 3280 section 4.1.2.5) field of certificate.
0x4 string	128 bytes	"SubjectKeyIdentifier"	Value of SubjectKeyIdentifier (RFC 3280 section 4.2.1.2) extension of certificate.
0x3 bin	4096 bytes	"RawPublicKey"	Raw value of public key associated with the certificate. (SubjectPublicKeyInfo->subjectPublicKey).
0x1 long	4 bytes	"PublicKeyLength"	Length (in bits) of the public key associated with the certificate

Data type	Maximum size of data	Data element name	Processing Rule or x.509 Certificate Field or Extension
			(SubjectPublicKeyInfo->subjectPublicKey).
0x4 string	254 bytes	"PublicKeyAlgorithm"	Name of the algorithm associated with the public key of the certificate (SubjectPublicKeyInfo->algorithm->algorithm).
0x3 bin	4096 bytes	"RawPublicKeyAlgorithmParameters"	Raw value of parameters associated with the public key of the certificate (SubjectPublicKeyInfo->algorithm->parameters).
0x4 string	8192 bytes	"DistinguishedName"	Subject (RFC 3280 section 4.1.2.6) field of certificate.
0x4 string	8192 bytes	"Country"	Country attribute of certificate Subject.
0x4 string	8192 bytes	"Organization"	Organization attribute of certificate Subject.
0x4 string	8192 bytes	"OrgUnit"	Organizational-unit attribute of certificate Subject.
0x10004 string indexed	8192 bytes	"CommonName"	Common name attribute of certificate Subject.
0x4 string	8192 bytes	"Locality"	Locality attribute of certificate Subject.
0x4 string	8192 bytes	"State"	State or province name attribute of certificate Subject.
0x4 string	8192 bytes	"Title"	Title attribute of certificate Subject.
0x4 string	8192 bytes	"GivenName"	Given name attribute of certificate Subject.
0x4 string	8192 bytes	"Initials"	Initials attribute of certificate Subject.
0x4 string	8192 bytes	"SurName"	Surname attribute of certificate Subject.
0x4 string	8192 bytes	"DomainComponent"	DomainComponent attribute of certificate Subject.
0x4 string	8192 bytes	"EMail"	RFC822 Name from Subject Alternative Name.
0x4 string	8192 bytes	"DeviceSerialNumber"	Serial number attribute of certificate Subject.

3.2.5.1.27 ICertAdminD::BackupGetDynamicFiles (Opnum 29)

The **BackupGetDynamicFiles** method retrieves the list of CA dynamic file names that need to be backed up. The dynamic files are those that are not included in the CA database backup and are created dynamically by the CA, for example: CRL files created by the CA. Note that **BackupOpenFile** and **BackupReadFile** cannot be used to open and read the dynamic files whose names are returned by this method. Dynamic files must be backed up by means outside this protocol.

```
HRESULT BackupGetDynamicFiles(
  [out, size_is(, *pcwcFiles)] WCHAR** ppwszzFiles,
  [out] LONG* pcwcFiles
);
```

ppwszzFiles: A pointer to a **WCHAR** pointer that will receive the list of NULL-terminated dynamic file names used by CA.

pcwcFiles: A pointer to the **LONG** value that specifies the number of characters in *ppwszzFiles*.

The CA Server MUST apply the following processing rule:

- The CA server MUST return a list of file names that are not part of database but which the CA deems necessary to be part of backup. An example of a CA dynamic file is the certificate revocation list (CRL). The files MUST be accessible to the CA. Each file name MUST be in UNC format. If no files are deemed necessary, the CA MUST set 0 as the value of *pcwcFiles parameter and return successfully.

3.2.5.1.28 ICertAdminD::RestoreGetDatabaseLocations (Opnum 30)

The **RestoreGetDatabaseLocation** method retrieves the list of CA Server database location names for all the database files being restored.

```
HRESULT RestoreGetDatabaseLocations(
  [out, size_is(, *pcwcPaths)] WCHAR** ppwszzDatabaseLocations,
  [out] LONG* pcwcPaths
);
```

ppwszzDatabaseLocations: A pointer to a **WCHAR** pointer that will receive the list of null-terminated database location names and log directory name. Detailed database file name structure formatting information is specified in section 2.2.4.

pcwcPaths: A pointer to the **LONG** value that specifies the number of bytes in *ppwszzDatabaseLocations*.

The CA Server MUST apply the following processing rule:

- The CA server MUST return a list that comprises the CA database file name and the locations to which the CA log and system files will be restored. The locations MUST be accessible to the CA. The database file name MUST be in UNC format and MUST be prefixed with "D". The log file location MUST be in UNC format and prefixed with a character whose value is 130. The system file location MUST be in UNC format and prefixed with a character whose value is 131.

3.2.5.2 Processing Rules for ICertAdminD2

The **ICertAdminD2** interface extends the **ICertAdminD** interface described in the preceding section.<53>

The version number for this interface is "1.0". The UUID (universally unique identifier) for this interface is: "7fe0d935-dda6-443f-85d0-1cfb58fe41dd".

Opnum values start with the value subsequent to the last opnum value in the last inherited method. Therefore, opnum for this interface starts with 31.

Methods in RPC Opnum Order

Method	Description
PublishCRLs	The **PublishCRLs** method forces a CA to publish CRLs and Delta CRLs. Opnum: 31
GetCAProperty	The **GetCAProperty** method is used to retrieve a given property's value from the CA. Opnum: 32
SetCAProperty	The **SetCAProperty** method is used to set CA properties. Opnum: 33
GetCAPropertyInfo	The **GetCAPropertyInfo** method is used to retrieve information about a property on the CA, such as its type and length. Opnum: 34
EnumViewColumnTable	The **EnumViewColumnTable** method retrieves information about a column from the CA database table. Opnum: 35
GetCASecurity	The **GetCASecurity** method is used to retrieve CA Server's Security Descriptor. Opnum: 36
SetCASecurity	The **SetCASecurity** method is used to set CA Server's Security Descriptor. Opnum: 37
Ping2	The **Ping2** method is used to determine if the CA service is started and responding. Opnum: 38
GetArchivedKey	The **GetArchivedKey** method is used to retrieve an archived private key and the associated certificate. Opnum: 39
GetAuditFilter	The **GetAuditFilter** method retrieves the list of events for which the CA server is currently set to create security audit events, as specified in [CIMC-PP]. Opnum: 40
SetAuditFilter	The **SetAuditFilter** method sets the list of events that the CA server MUST create security audit events, as specified in [CIMC-PP]. Opnum: 41

Method	Description
GetOfficerRights	The **GetOfficerRights** method is used to retrieve the officer rights, as specified in [CIMC-PP]. Opnum: 42
SetOfficerRights	The **SetOfficerRights** method is used to set the officer rights. Opnum: 43
GetConfigEntry	The **GetConfigEntry** method retrieves the CA configuration data. Opnum: 44
SetConfigEntry	The **SetConfigEntry** method is used to set the configuration information for the Certificate Server. Opnum: 45
ImportKey	The **ImportKey** method adds an encrypted key set to an item in the CA database. Opnum: 46
GetMyRoles	The **GetMyRoles** method retrieves the roles, as specified in [CIMC-PP], assigned to the user who calls the method. Opnum: 47
DeleteRow	The **DeleteRow** method deletes a row or set of rows from a database table. Opnum: 48

All methods MUST NOT throw exceptions.

3.2.5.2.1 ICertAdminD2::PublishCRLs (Opnum 31)

The **PublishCRLs** method instructs a CA to publish CRLs and Delta CRLs. This call can either cause the republishing of the current CRLs or cause the CA to create and publish new CRLs.

```
HRESULT PublishCRLs(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] FILETIME FileTime,
  [in] DWORD Flags
);
```

pwszAuthority: See pwszAuthority definition in section 3.2.5.1.1.

FileTime: Contains a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). Specifies the nextUpdate value of the CRL as specified in [RFC3280] section 5.1.2.5 in Coordinated Universal Time (Greenwich Mean Time).

Flags: An unsigned integer value that specifies the type of CRLs to publish and the publishing parameters. This parameter MUST be set to a combination of the following values:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
F	D	0	0	B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Value	Description
B	If 1, CA MUST publish Base CRL.
D	If 1, CA MUST publish Delta CRL.
F	If 1, CA MUST republish the existing CRLs.

The CA Server MUST apply the following processing rules.

1. If the CA has not implemented revocation via CRL, the CA MUST return an error when this method is invoked. Otherwise, if the CA has implemented revocation via CRL, the CA MUST continue with the following processing rules when this method is invoked.

2. If the F bit is set in *Flags*, the *FileTime* parameter is ignored and the following MUST occur.

   - If the B bit is set in *Flags*, the CA MUST publish the most recent Base CRL for each valid CA key to such locations as have been identified in Config.CA.CDP.Publish.To.<54>

   - If the D bit is set in *Flags*, the CA MUST publish the most recent Delta CRL for each valid CA key to such locations as have been identified in Config.CA.CDP.Publish.To.<55>

   - After publishing the existing CRL or CRLs, the CA SHOULD update the following CRL table data elements in the CRL table entry for each CRL that was published:

     - CRL.Publish.Status.Code

     - CRL.Publish.Error

     - CRL.LastPublished (for a base CRL only): The CA SHOULD set this data element to the current time minus the lifetime of the current base CRL.

     - CRL.Publish.Attempts: The CA SHOULD increment this data element by one to indicate the additional publish attempt for the existing CRL

3. If the F bit is NOT set in *Flags*, the following MUST occur.

   1. The CA MUST create a CRL for each valid CA key.

      - If the B bit is set in *Flags*, the type of CRL that the CA creates for each valid CA key MUST be a new base CRL.

      - If the D bit is set in *Flags*, the type of CRL that the CA creates for each valid CA key MUST be a new Delta CRL.

   2. The CA SHOULD check the Request table to determine which certificates to include in a CRL, according to the following criteria.

      - If Request.Revocation.Date is NOT NULL and is in the past:

- If the Request.Disposition is not "certificate revoked", do not include the certificate's Serial.Number on the CRL.
  - One implication of this rule is that if a previously revoked certificate is unrevoked (for example, when the certificate is removed from certificate hold) and if it expires between the times that two successive regularly published CRLs are published, then the certificate's serial number does not appear on the latter CRL. This outcome violates [RFC3280] section 3.3 "An entry MUST NOT be removed from the CRL until it appears on one regularly scheduled CRL issued beyond the revoked certificate's validity period."
- If the Request.Disposition is "certificate revoked":
  - For a delta CRL, exclude all certificates whose Request.Revoked.When is before the CRL.This.Update of the oldest unexpired base CRL.
  - If the Publish.Expired.Cert.In.CRL column is set to 1, the certificate's Serial.Number should be included in CRLs regardless of the certificate's notAfter time.
  - If the Publish.Expired.Cert.In.CRL column is not set to 1:
    - If the certificate's notAfter time is before the CRL.This.Publish of the last CRL, do not include the certificate Serial.Number on the CRL.
    - If the certificate's notAfter time is after the CRL.This.Publish of the last CRL, include the certificate's Serial.Number on the CRL.

3. At CRL creation time, the CA SHOULD create a new CRL table entry, populating the following data elements:
   - CRL.Number
   - CRL.Name.Id
   - CRL.Min.Base (for a delta CRL)
   - CRL.Count
   - CRL.This.Publish
   - CRL.This.Update : The CA SHOULD use the current time minus Config.CA.Clock.Skew.Minutes <56>. The value of CRL.This.Update MUST NOT be less than the NotBefore date of the current CA certificate. If the calculated value of CRL.This.Update is less than the NotBefore date of the current CA certificate is, the CA SHOULD replace the CRL.This.Update value with the NotBefore date of the current CA certificate.
   - CRL.Next.Publish: The CA SHOULD set this data element to the current time plus Config.Base.CRL.Validity.Period for a base CRL or Config.Delta.CRL.Validity.Period, for a delta CRL.
   - CRL.Effective (for delta CRL)
   - CRL.Propagation.Complete: The CA SHOULD set this data element to the current time plus overlap period.<57>
4. If a FileTime is provided, which is indicated by a nonzero FileTime value, the CA MUST use it in following manner:

1. If the FileTime value is less than or equal to the current time, the CA MUST fail with an error code ERROR_INVALID_PARAMETER.

2. Otherwise, the CA MUST use the value supplied in the FileTime parameter as the CRL.Next.Update.

5. If a FileTime is not provided, the CA SHOULD set CRL.Next.Update to the current time plus Config.Base.CRL.Validity.Period for a base CRL or Config.Delta.CRL.Validity.Period, for a delta CRL, plus an overlap period<58><59>.

6. If the CRL.Next.Update value is after the NotAfter date of the current CA certificate, the CA SHOULD replace the CRL.Next.Update value with the NotAfter date of the current CA certificate.

7. After any required new CRL table entries are created, the CA MUST create and publish a CRL for each valid CA key to such locations as have been identified in Config.CA.CDP.Publish.To<60>. The CA SHOULD allocate data to the various CRL extensions as follows:

   - CRL.This.Update SHOULD be used for the thisUpdate field of the CRL.
   - CRL.Next.Update SHOULD be used for the nextUpdate field of the CRL.
   - For each certificate determined in rule 3.2 above to be included on the CRL, Serial.Number and Request.Revocation.Date SHOULD be used respectively as the userCertificate and revocationDate components of the revoked certificates list in the CRL.
   - CRL.Next.Publish SHOULD be used for the nextPublish extension of the CRL.
   - CRL.Number SHOULD be used for the CRL Number extension of the CRL.
   - CRL.Min.Base SHOULD be used for the Delta CRL indicator extension of the CRL (for a delta CRL only).
   - Otherwise, CRL fields and extensions are as specified in [RFC3280] section 5.

8. After publishing, the CA SHOULD update the following CRL table data elements:

   - CRL.Publish.Status.Code
   - CRL.Publish.Error
   - CRL.LastPublished (for a base CRL only): The CA SHOULD set this data element to the current time minus the lifetime of the current base CRL.
   - CRL.Publish.Attempts: The CA SHOULD set this data element to one to indicate the publish attempt for the newly created CRL.

3.2.5.2.2 ICertAdminD2::GetCAProperty (Opnum 32)

The **GetCAProperty** method is used to retrieve a given property's value from the CA.

```
HRESULT GetCAProperty(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] LONG PropId,
  [in] LONG PropIndex,
  [in] LONG PropType,
```

```
  [out, ref] CERTTRANSBLOB* pctbPropertyValue
);
```

pwszAuthority: See pwszAuthority definition in section 3.2.5.1.1.

PropId: An integer value specifying the property to be returned. The use of *PropIds* is as specified in [MS-WCCE] section 3.2.4.2.2. The *PropID* value must be one of the values in the table labeled PropId in [MS-WCCE] section **3.2.4.1.2.2**.

PropIndex: Some of these properties (the ones labeled as "indexed" in the table in [MS-WCCE] section **3.2.4.1.2.2**) have arrays of values. This parameter MUST be used as the index into such an array. For properties that are not arrays, this parameter MUST be ignored.

PropType: An integer value that specifies the property data type.

Value	Meaning
PROPTYPE_LONG 0x00000001	The property type is a signed long integer or a byte array.
PROPTYPE_BINARY 0x00000003	The property type is binary data.
PROPTYPE_STRING 0x00000004	The property type is a Unicode string.

pctbPropertyValue: If the function succeeds, this method MUST return a **CERTTRANSBLOB** structure that contains the property value. If the function fails, the contents are undefined.

Note The numeric values for the constants listed in this topic are defined in the table for the *PropID* parameter.

The data type of the value returned depends on the value specified in the *PropType* parameter and the property specified in the *PropID* parameter.

- If PROPTYPE_STRING is specified in the *PropType* parameter, pctbPropertyValue MUST be a pointer to a **CERTTRANSBLOB** structure. The **pb** member of the structure points to the little-endian encoded Unicode string. The length, in bytes, of the string MUST be contained in the cb member.
- If PROPTYPE_LONG is specified in the *PropType* parameter, there are two possible return types depending on the *PropID*. The first type is the return of a CAINFO structure (as specified in [MS-WCCE] section **2.2.2.3**) and the second type is for the return of a BYTE array.
  - If the value passed in PropId maps to one of the properties listed below, *pctbPropertyValue* is a pointer to a **CERTTRANSBLOB** structure, and the **pb** member of that structure MUST contain a pointer to a CAINFO structure that contains the values of the properties listed as follows. The marshaling rules for a CAINFO structure in a CERTTRANSBLOB are specified in [MS-WCCE] section 2.2.2.1.5:
    - CR_PROP_CATYPE
    - CR_PROP_CASIGCERTCOUNT
    - CR_PROP_CAXCHGCERTCOUNT

- CR_PROP_EXITCOUNT
- CR_PROP_CAPROPIDMAX
- CR_PROP_KRACERTUSEDCOUNT
- CR_PROP_ROLESEPARATIONENABLED
- CR_PROP_KRACERTCOUNT
- CR_PROP_ADVANCEDSERVER

- If the value passed in *PropId* maps to one of the properties listed below, *pctbPropertyValue* is a pointer to a **CERTTRANSBLOB** structure, and the **pb** member of the structure points to a byte array containing the value for the requested property. The marshaling rules for each property are specified in the subsection of [MS-WCCE] section **3.2.1.4.2.2** that corresponds to the property name. The **cb** member contains the length of the byte array.

  - CR_PROP_CACERTSTATE
  - CR_PROP_CRLSTATE
  - CR_PROP_KRACERTSTATE
  - CR_PROP_BASECRLPUBLISHSTATE
  - CR_PROP_DELTACRLPUBLISHSTATE
  - CR_PROP_CACERTSTATUSCODE
  - CR_PROP_CAFORWARDCROSSCERTSTATE
  - CR_PROP_CABACKWARDCROSSCERTSTATE

- If PROPTYPE_BINARY is specified in the *PropType* parameter, *pctbPropertyValue* is MUST be a pointer to a **CERTTRANSBLOB** structure. The pb member of the structure points to the requested binary large object (BLOB).

  Based on the property identifier passed in *PropId*, the binary data pointed to by the *pb* member MUST be formatted as follows:

  - CR_PROP_CASIGCERT: MUST be an X.509 certificate encoded using Distinguished Encoding Rules (DER), as specified in [X660].
  - CR_PROP_BASECRL: MUST be a X.509 CRL encoded using Distinguished Encoding Rules (DER), as specified in [X660].
  - CR_PROP_CAFORWARDCROSSCERT: MUST be a X.509 certificate encoded using Distinguished Encoding Rules (DER), as specified in [X660].
  - CR_PROP_CABACKWARDCROSSCERT: MUST be a X.509 certificate encoded using Distinguished Encoding Rules (DER), as specified in [X660].
  - CR_PROP_CAXCHGCERT: MUST be a X.509 certificate encoded using Distinguished Encoding Rules (DER), as specified in [X660].
  - CR_PROP_CASIGCERTCHAIN: MUST be a CMS message [RFC2797] encoded using Distinguished Encoding Rules (DER). [X660].

- CR_PROP_CASIGCERTCRLCHAIN: MUST be a CMS message, as specified in [RFC2797], encoded using Distinguished Encoding Rules (DER), as specified in [X660].
- CR_PROP_CASIGCERTCRLCHAIN: MUST be a CMS message, as specified in [RFC2797], encoded using Distinguished Encoding Rules (DER), as specified in [X660].
- CR_PROP_CAXCHGCERTCRLCHAIN: CR_PROP_CASIGCERTCRLCHAIN: MUST be a CMS message, as specified in [RFC2797], encoded using Distinguished Encoding Rules (DER), as specified in [X660].
- CR_PROP_DELTACRL: MUST be a X.509 CRL encoded using Distinguished Encoding Rules (DER) [X660].
- CR_PROP_KRACERT: MUST be a X.509 CRL encoded using Distinguished Encoding Rules (DER), as specified in [X660].

The marshaling rules for each of the above properties into a CERTTRANSBLOB are specified in [MS-WCCE] sections 2.2.2.1.2 (for X.509 certificate), 2.2.2.1.3 (X.509 CRL), and 2.2.2.1.4 (CMS message).

3.2.5.2.3 ICertAdminD2::SetCAProperty (Opnum 33)

The **SetCAProperty** method is used to set CA properties.

```
HRESULT SetCAProperty(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] LONG PropId,
  [in] LONG PropIndex,
  [in] LONG PropType,
  [in] CERTTRANSBLOB* pctbPropertyValue
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

PropId: A **LONG** value that specifies one and exactly one of the following property identifiers. The use of PropIds, is as specified in [MS-WCCE] section 3.2.4.2.2.

Value	Meaning
0x0000001a	A binary object that contains the CA's **key recovery agent (KRA)** certificate to be added at the index specified by *PropIndex* parameter.
0x00000019	The maximum number of KRA certificates available on the certification authority.
0x00000018	The minimum number of KRAs to use when archiving a private key. For more information on KRA usage, see [MSFT-ARCHIVE].
0x0000001d	A collection of name and object identifier (OID) pairs that identify the templates supported by a CA.

PropIndex: A **LONG** value for the index of the key recovery agent (KRA) certificate to set when the provided PropId is 0x1a. For other *PropId* values, it MUST be 0.

PropType: A **LONG** value that specifies the type of the property. This parameter MUST be one of the following values.

Value	Meaning
PROPTYPE_LONG 0x00000001	Signed LONG data
PROPTYPE_BINARY 0x00000003	Binary data
PROPTYPE_STRING 0x00000004	Unicode String data

pctbPropertyValue: A pointer to **CERTTRANSBLOB** that specifies the new property value. The format for the value contained in **CERTTRANSBLOB** is specific to the propId as defined below:

Value of PropID	Format for values in CERTTRANSBLOB
CR_PROP_KRACERTUSEDCOUNT 0x00000018	**pb** member of **CERTTRANSBLOB** MUST point to unsigned integer value (little-endian format) and **cb** member of **CERTTRANSBLOB** MUST contain the length of the bytes containing the value.
CR_PROP_KRACERTCOUNT 0x00000019	**pb** member of **CERTTRANSBLOB** MUST point to unsigned integer value (little-endian format) and **cb** member of **CERTTRANSBLOB** MUST contain the length of the bytes containing the value.
CR_PROP_KRACERT 0x0000001a	**pb** member of **CERTTRANSBLOB** MUST point to ASN.1 DER (as specified in [ITUX690]) encoded byte array of **Certificate. cb** member of **CERTTRANSBLOB** MUST contain the length of the array.
CR_PROP_TEMPLATES 0x0000001d	As specified in [MS-WCCE] section 3.3.29.

The table below defines the values that MUST be set for PropIndex and PropType for each one of the property values passed via *PropID*.

PropID value	PropIndex MUST be	PropType MUST be
0x0000001a	The minimum index is 0.	0x00000003
0x00000019	0x00000000	0x00000001
0x00000018	0x00000000	0x00000001
0x0000001d	0x00000000	0x00000004

When processing the **SetCAProperty** method, the server determines its behavior based on the requested property ID (*PropID* parameter). All valid property IDs are listed in the table above. The CA MUST return the error value ERROR_INVALID_PARAMETER if any of the conditions below are met:

- The value of *PropID* is not listed in the table above OR
- For a given *PropID* value the *PropIndex* value does not match the required values defined in the table above OR

- For a given *PropID* value the *PropType* value does not match the required values defined in the table above

The CA server MUST use the property values to modify the data (as specified in Abstract Data Model in [MS-WCCE] section 3.2.1) maintained by CA as part of configuration.

The CA server MUST apply the following processing rules:

1. The value of CR_PROP_KRACERTUSEDCOUNT MUST be between 1 and the current configured value of CR_PROP_KRACERTCOUNT property. The initial value for CR_PROP_KRACERTCOUNT property MUST be 0.

2. The value of CR_PROP_KRACERTCOUNT MUST be less than the current value of CR_PROP_KRACERTCOUNT.

3. The CA must increase the value of the property CR_PROP_KRACERTCOUNT by one each time **SetCAProperty** (CR_PROP_KRACERT) is called to with a certificate that does not already exist on the CA.

3.2.5.2.4 ICertAdminD2::GetCAPropertyInfo (Opnum 34)

The **GetCAPropertyInfo** method used to retrieve information about a property on the CA, such as its type and length.

```
HRESULT GetCAPropertyInfo(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out] LONG* pcProperty,
  [out, ref] CERTTRANSBLOB* pctbPropInfo
);
```

pwszAuthority: See *pwszAuthority* definition in **ICertAdminD::SetExtension**.

pcProperty: An integer value containing the number of property structures returned.

pctbPropInfo: A **CERTTRANSBLOB** structure containing zero or more **CATRANSPROP** structures. For more information on **CERTTRANSBLOB** and **CATRANSPROP** structures, see Common Structures.

ICertAdminD2::**GetCAPropertyInfo** is as specified in [MS-WCCE] section 3.2.4.1.2.3.

3.2.5.2.5 ICertAdminD2::EnumViewColumnTable (Opnum 35)

The **EnumViewColumnTable** method retrieves information about a column from the CA database table.

```
HRESULT EnumViewColumnTable(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD iTable,
  [in] DWORD iColumn,
  [in] DWORD cColumn,
  [out] DWORD* pcColumn,
  [out, ref] CERTTRANSBLOB* pctbColumnInfo
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

iTable: An unsigned integer that specifies the database table to be used for the enumeration. This MUST be set from the following values:

Value	Meaning
0x00000000	Request table
0x00003000	Extensions table
0x00004000	Attributes table
0x00005000	CRL table

iColumn: An unsigned integer that specifies the column number to be enumerated. Valid values are from 0 to one less than the maximum number of columns for the table.

cColumn: An unsigned integer that specifies request count of columns to return.

pcColumn: A pointer to unsigned integer that receives the returned count of **CERTTRANSDBCOLUMN** structures.

pctbColumnInfo: A pointer to a **CERTTRANSBLOB** structure. Upon return, pb points to an array of the marshaled **CERTTRANSDBCOLUMN** structures. The format and marshaling for value of *pctbColumnInfo* MUST be as specified in section 2.2.1.7.

The **EnumViewColumnTable** method returns information about columns associated with a specific table to the client.

The CA server MUST enforce the following processing rules:

- The CA Server MUST enforce that *iTable* parameter has value as specified in the table previous, otherwise it MUST fail with error ERROR_INVALID_PARAMETER.
- The CA Server MUST enforce that *iColumn* is less than the number of columns associated with the table. Otherwise it MUST fail with error ERROR_ARITHMETIC_OVERFLOW.
- The CA Server MUST enforce that *cColumn* is greater than 0, otherwise it MUST fail with error ERROR_INVALID_PARAMETER<61>.
- The CA server MUST use value of iColumn to identify the column index associated with the table (identified by value of *iTable* parameter).
- The number of column information returned MUST be a minimum of **cColumn** value and the remaining number of columns in the table (starting from **iColumn**). The value of *pcColumn MUST be set to the number of the column information returned.

3.2.5.2.6 ICertAdminD2::GetCASecurity (Opnum 36)

The **GetCASecurity** method is used to retrieve CA security, as defined in Abstract Data Model (section 3.2.1).

```
HRESULT GetCASecurity(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out, ref] CERTTRANSBLOB* pctbSD
```

```
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

pctbSD: A pointer to a **CERTTRANSBLOB** data structure that contains the CA's security descriptor. Security descriptors are specified in [MS-DTYP] section 2.4.6.

3.2.5.2.7 ICertAdminD2::SetCASecurity (Opnum 37)

The **SetCASecurity** method is used to set the CA security, as defined in the ADM.

```
HRESULT SetCASecurity(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, ref] CERTTRANSBLOB* pctbSD
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

pctbSD: A pointer to a **CERTTRANSBLOB** data structure that holds the security descriptor. Security descriptors are specified in [MS-DTYP] section **2.4.6**.

The CA SHOULD use the permissions set in **pctbSD** to deny and allow permissions to CA functionality.<62>

3.2.5.2.8 ICertAdminD2::Ping2 (Opnum 38)

The **Ping2** method is used to determine if the CA service is started and responding.

```
HRESULT Ping2(
  [in, string, unique] wchar_t const* pwszAuthority
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

ICertAdminD2::Ping2 is as specified in [MS-WCCE] section 3.2.4.1.2.3.

3.2.5.2.9 ICertAdminD2::GetArchivedKey (Opnum 39)

The **GetArchivedKey** method is used to retrieve an archived private key and the associated certificate.

```
HRESULT GetArchivedKey(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId,
  [out, ref] CERTTRANSBLOB* pctbArchivedKey
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

dwRequestId: An unsigned integer value that specifies the RequestId of the certificate request for which the archived private key and associated certificate is being requested.

pctbArchivedKey: A pointer to a **CERTTRANSBLOB** structure that MUST contain, on successful response, the archived private key and associated certificate.

ArchivedKey Property Value Processing and Format

The CA server MUST create the ArchivedKey property value by using the following processing rules:

1. Server MUST construct an enveloped PKCS #7 with the following requirement.
   - RecipientInfos: This field MUST reference the KRA certificate (for which the CA server is configured) that MUST be used to encrypt the client private key. The exact format of RecipientInfos is as specified in [RFC2315] section 10.2.
   - EncryptedContent: This field MUST include the encrypted private key associated with the certificate request.
2. Server MUST construct a PKCS #7 with the following requirement:
   - ContentType: This field MUST be SignedData (1.2.840.113549.1.7.2 PKCS#7 Signed).
   - Content: This field MUST be a SignedData with the following values for its fields.
   - ContentInfo: This field MUST have the following values for its fields.
   - ContentType This field MUST be Data (1.2.840.113549.1.7.1 PKCS#7 Data).
   - Content: This field MUST be the ASN.1 DER encoded enveloped PKCS #7 constructed in step 1 above.
   - Certificates: This field MUST include the current CA signing certificate that is used to verify this PKCS#7 message.
   - SignerInfos: The first SignerInfo in the SignerInfos collection MUST use the key associated with the current CA signing certificate.
3. The ASN.1 DER encoded PKCS#7 signed data created in step 2 MUST be the value of the ArchivedKey property in the RequestTable (see section 3.2.1).

The **GetArchivedKey** method is used to retrieve the archived private key and issued certificate from the CA's database.

The CA server MUST enforce the following processing rules:

- The CA MUST look up the request based on the provided *dwRequestId* parameter in the CA database RequestTable.
- If the request is not found, the CA MUST fail the request with error ERROR_REQUESTED_PROPERTYVALUE_EMPTY.
- If the request is found, the CA MUST ensure that value of the Request.Disposition column in the identified row is "certificate issued".
- The CA MUST also ensure that the request identified has ArchivedKey property value, otherwise the CA Server MUST fail with error ERROR_INVALID_PARAMETER.
- The **pb** field of the *pctbArchivedBlob* parameter MUST reference the value of the ArchivedKey property, and the **cb** field of the *pctbArchivedBlob* parameter MUST contain the length of ArchivedKey property value in bytes.

3.2.5.2.10 ICertAdminD2::GetAuditFilter (Opnum 40)

The **GetAuditFilter** method retrieves the list of events for which the CA server is currently set to create security audit events, as specified in [CIMC-PP].

```
HRESULT GetAuditFilter(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out] DWORD* pdwFilter
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

pdwFilter: An unsigned integer that specifies the current audit settings. This can be a bitwise-OR combination of zero or more of the following values.

Value	Meaning
0x00000001	Audit CA Server for following events: ▪ ServerControl ▪ Registration of ICertAdminD interface. ▪ Unregistration of ICertAdminD interface.
0x00000002	Audit CA Server for following method calls: ▪ BackupPrepare ▪ BackupEnd
0x00000004	Audit CA Server for following method calls: ▪ ICertRequestD::Request ▪ ResubmitRequest ▪ DenyRequest ▪ SetAttributes ▪ SetExtensions ▪ ImportCertificate ▪ DeleteRow
0x00000008	Audit CA Server for following method calls: ▪ RevokeCertificate ▪ PublishCRL ▪ PublishCRLs

Value	Meaning
0x00000010	Audit CA Server for following method calls: ▪ SetCASecurity ▪ SetOfficerRights ▪
0x00000020	Audit CA Server for following method calls: ▪ GetArchivedKey ▪ ImportKey
0x00000040	Audit CA Server for following method calls: ▪ SetCAProperty ▪ SetConfigEntry

The **GetAuditFilter** method is used to retrieve the audit filter currently in use (initialize to 0 during the registration of the interfaces and can be modified by call to **SetAuditFilter** method).

3.2.5.2.11 ICertAdminD2::SetAuditFilter (Opnum 41)

The **SetAuditFilter** method sets the list of events that the CA server MUST create security audit events, as specified in [CIMC-PP].

```
HRESULT SetAuditFilter(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwFilter
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

dwFilter: An unsigned integer that specifies the events to be audited by the CA. For possible values, see section 3.2.5.2.10.

The **SetAuditFilter** method is used to set the audit filter value passed in by the client. The audit filter value is used to determine what actions will get audited.

The CA Server MUST start auditing the methods based on the value of dwFilter parameter. The list of methods that MUST be audited for the value is specified in section 3.2.5.2.10.

3.2.5.2.12 ICertAdminD2::GetOfficerRights (Opnum 42)

The **GetOfficerRights** method is used to retrieve the officer rights, as specified in [CIMC-PP].

```
HRESULT GetOfficerRights(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out] BOOL* pfEnabled,
```

```
  [out, ref] CERTTRANSBLOB* pctbSD
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

pfEnabled: A pointer to a Boolean value.

pctbSD: A pointer to the **CERTTRANSBLOB** structure that contains the marshaled information specified in section 2.2.1.11.1.

The following processing rules apply:

1. If the CA server does not support Enrollment Agent rights:
    1. If no Officer rights are configured, the server MUST set the value of **pfEnabled* to 0, the *pb* member of *pctbSD* to NULL, and the *cb* member to 0.
    2. If Officer rights are configured on the CA Server, it MUST set the value of **pfEnabled* to nonzero and return the marshaled data specified in section 2.2.1.11.1 in pctbSD.
2. If the CA server supports Enrollment Agent rights:
    1. If no officer rights are configured (Config.Permissions.Officer.Rights) and no Enrollment Agent rights (Config.Permissions.Enrollment.Agent.Rights) are configured on the CA server, then the server MUST set the value of *pfEnabled to 0 and pb member of pctbSD MUST contain the marshaled data specified in section 2.2.1.11.1.
    2. If no officer rights are configured, but Enrollment Agent rights are configured on the CA server, then the server MUST set the value of *pfEnabled to 0 and pctbSD MUST contain the marshaled data specified in section 2.2.1.11.1.
    3. If Officer rights are configured, but no Enrollment Agent rights are configured on the CA server, then the server MUST set the value of *pfEnabled to nonzero and pb member of pctbSD MUST contain the marshaled data specified in section 2.2.1.11.1.
    4. If Officer rights are configured and Enrollment Agent rights are configured on the CA server, then the server MUST set the value of *pfEnabled to nonzero and pctbSD MUST contain the marshaled data specified in section 2.2.1.11.1.

3.2.5.2.13 ICertAdminD2::SetOfficerRights (Opnum 43)

The **SetOfficerRights** method is used to set Officer rights or Enrollment Agent rights. Information on **role separation** is specified in [CIMC-PP].

```
HRESULT SetOfficerRights(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] BOOL fEnable,
  [in, ref] CERTTRANSBLOB* pctbSD
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

fEnable: A **BOOL** parameter. This parameter MUST be set to one of the following values:

	3										2									1									
1	**0**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

F - fRightsEnable

If bits 0 through 15 are 0, then disable access rights (**Officer** or **Enrollment Agent**) and ignore the value of *pctbSD*.

R - RightsType

If bits 16 through 31 are 0, then the security descriptor in the *pctbSD* parameter is for **Officer Rights**.

If bits 16 through 31 are nonzero, the security descriptor in the *pctbSD* is for the **enrollment agents**.

pctbSD: A pointer to the **CERTTRANSBLOB** structure that holds the marshaled security descriptor, as specified in [MS-DTYP] section 2.4.6.

The following processing rules apply.

1. If fRightsEnable is 0 and RightsType is 0, the server MUST disable Enrollment Agent access rights (Config.Permissions.Enrollment.Agent.Rights) and ignore the value of *pctbSD*.
2. If fRightsEnable is 0 and RightsType is nonzero, the server MUST disable Officer access rights (Config.Permissions.Officer.Rights) and ignore the value of *pctbSD*.
3. If fRightsEnable is nonzero and RightsType is 0, the server MUST set the security descriptor specified in *pctbSD* as Officer access rights.
4. If fRightsEnable is nonzero and RightsType is nonzero, the server MUST set the security descriptor specified in *pctbSD* as Enrollment Agent access rights.

3.2.5.2.14 ICertAdminD2::GetConfigEntry (Opnum 44)

The **GetConfigEntry** method retrieves the CA configuration data. Configuration data is represented as a hierarchical data structure with the following format: [\pwszAuthority][\pwszNodePath][\pwszEntry].

```
HRESULT GetConfigEntry(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, string, unique] wchar_t const* pwszNodePath,
  [in, string, ref] wchar_t const* pwszEntry,
  [out, ref] VARIANT* pVariant
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

pwszNodePath: A string value that represents the node path for the configuration information. This parameter can be an EMPTY string, and MUST NOT be NULL.

pwszEntry: A string value that represents the name of the leaf entry whose information is being retrieved. This value can be an EMPTY string, and MUST NOT be NULL.

pVariant: A pointer to a VARIANT that receives the requested information.

The **GetConfigEntry** method retrieves the CA configuration data or configuration data hierarchy information.

The following processing rules apply:

1. If *pwszAuthority* parameter is EMPTY and *pwszNodePath* parameter is EMPTY and *pwszNodeEntry* is EMPTY, then the CA MUST return all available leaf properties' names that exist in the configuration's root node as a VARIANT array.

2. If *pwszAuthority* is EMPTY and *pwszNodePath* is EMPTY and *pwszNodeEntry* is not EMPTY, then the CA must return the leaf property value identified by *pwszNodeEntry* that exists under the Configuration root node as a VARIANT.

3. If *pwszAuthority* is EMPTY and *pwszNodePath* is not EMPTY, then for any value of *pwszNodeEntry* the CA MUST fail the call with an error code of 0x80070057.

4. If *pwszAuthority* parameter is not EMPTY and *pwszNodePath* is EMPTY and *pwszNodeEntry* is EMPTY, the CA MUST return all available leaf properties' names that exist under the *pwszAuthority* node as a VARIANT array.

5. If *pwszAuthority* parameter is not EMPTY and *pwszNodePath* is EMPTY and *pwszNodeEntry* is not EMPTY, the CA MUST return the leaf property value identified by *pwszNodeEntry* that exists under the *pwszAuthority* node as a VARIANT array.

6. If *pwszAuthority* parameter is not EMPTY and *pwszNodePath* is not EMPTY and *pwszEntry* is EMPTY, the CA MUST return all available leaf properties' names that exist under the *pwszNodePath* node as a VARIANT array.

7. If *pwszAuthority* parameter is not EMPTY and *pwszNodePath* is not EMPTY and *pwszEntry* is not EMPTY, the CA MUST return the leaf property value identified by *pwszNodeEntry* that exists under the *pwszNodePath* node as a VARIANT array.

3.2.5.2.15 ICertAdminD2::SetConfigEntry (Opnum 45)

The **SetConfigEntry** method is used to set the configuration information for the certificate server.

```
HRESULT SetConfigEntry(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in, string, unique] wchar_t const* pwszNodePath,
  [in, string, ref] wchar_t const* pwszEntry,
  [in, ref] VARIANT* pVariant
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

pwszNodePath: A string value that represents the node path for the configuration information. This parameter can be an EMPTY string, and MUST NOT be NULL.

pwszEntry: A string value that represents the name of the leaf entry whose information is being set. This value can be an EMPTY string, and MUST NOT be NULL.

pVariant: A pointer to VARIANT that specifies the information to set. If this value is EMPTY, the indicated entry MUST be deleted.

The following processing rules apply:

1. If all arguments are provided, the CA MUST update the configuration with the value provided.
2. If the configuration value parameter passed in is empty, the indicated configuration entry MUST be deleted.

3.2.5.2.16 ICertAdminD2::ImportKey (Opnum 46)

The **ImportKey** method adds an encrypted key set to an item in the CA database.

```
HRESULT ImportKey(
  [in, string, unique] wchar_t const* pwszAuthority,
  [in] DWORD dwRequestId,
  [in, string, unique] wchar_t const* pwszCertHash,
  [in] DWORD dwFlags,
  [in, ref] CERTTRANSBLOB* pctbKey
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

dwRequestId: An unsigned integer value that represents the certificate request ID in the CA database. If the certificate hash (passed in as pwszCertHash) is to be used instead of the request ID the parameter MUST be 0. MUST be ignored on receipt.

pwszCertHash: A null-terminated UNICODE string value that represents the SHA1 hash of the ASN.1 DER encoded (as specified in [X660]) certificate data, formatted as hexadecimal string. For pwszCertHash to be used, a value of zero for dwRequestId MUST be specified.

dwFlags: An unsigned integer that specifies the optional flags for this method.

Value	Meaning
0x00010000	Overwrite existing archived key if present.

pctbKey: A **CERTTRANSBLOB** structure containing the ASN.1 DER (as specified in [X660] and [X690]) encoded PKCS#7 message (as specified in [RFC2315]) containing the private key to be archived. Content of enveloped PKCS#7 is as specified in [MS-WCCE] section 3.2.4.1.1.1.4.4.

The following processing rules apply:

1. The CA MUST process the enveloped PKCS#7 in pctbKey parameter as specified in [MS-WCCE] section 3.2.4.1.1.1.4.4.
2. If Request ID is not 0, then the CA MUST look up the request based on the provided *dwRequestId* parameter in the CA database RequestTable.
    1. If the request is not found, the method MUST fail with error ERROR_INVALID_PARAMETER.

2. If the request is found, then the CA MUST verify that the private key (decrypted in step 1) is cryptographically related to the public key in the private key. If the keys are not related, then the method MUST fail with error ERROR_INVALID_PARAMETER.

3. If Request ID is 0, then *pwszCertHash* MUST be non-null; otherwise, the CA server MUST fail with error ERROR_INVALID_PARAMETER.

4. If Request ID is 0 and *pwszCertHash* is non-null, the CA MUST look up the request based on the provided *pwszCertHash* parameter value in the CA RequestTable by computing the SHA1 hash of each issued certificate in the RequestTable and comparing the hexadecimal string form of it with the value specified in *pwszCertHash*.

   1. If the request is not found, the CA Server MUST fail the request with error ERROR_INVALID_PARAMETER.

   2. If the request is found, then the CA MUST verify that the private key (decrypted in step 1) is cryptographically related to the public key in the certificate. If the keys are not related, then the method MUST fail with error ERROR_INVALID_PARAMETER.

5. If the request is found, has an encrypted private key associated with it, and the value of *dwFlags* is not 0x00010000, then the CA MUST fail with error ERROR_INVALID_PARAMETER.

6. If the request is found, has an encrypted private key associated with it, and the value of *dwFlags* is 0x00010000, then the CA MUST replace the encrypted private key (in the request stored in the RequestTable) with the encrypted private key specified in *pctbKey* parameter.

7. If the request is found and does not have an encrypted private key associated with it, then the CA MUST replace the encrypted private key (in the request stored in the RequestTable) with the encrypted private key specified in *pctbKey* parameter.

3.2.5.2.17 ICertAdminD2::GetMyRoles (Opnum 47)

The **GetMyRoles** method retrieves the CA roles, as specified in [CIMC-PP], assigned to the user who calls the method.

```
HRESULT GetMyRoles(
  [in, string, unique] wchar_t const* pwszAuthority,
  [out] LONG* pdwRoles
);
```

pwszAuthority: See the *pwszAuthority* definition in section 3.2.5.1.1.

pdwRoles: A signed integer value that represents the retrieved CA roles for the caller. This can be a bitwise-OR combination of zero or more of CA security values as defined in the ADM and based on CA implementation.<63>

For *pdwRoles*, the server MUST return a signed integer that represents the CA security as defined in the ADM.

3.2.5.2.18 ICertAdminD2::DeleteRow (Opnum 48)

The **DeleteRow** method deletes a row or set of rows from a database table.

```
HRESULT DeleteRow(
  [in, string, unique] wchar_t const* pwszAuthority,
```

```
  [in] DWORD dwFlags,
  [in] FILETIME FileTime,
  [in] DWORD dwTable,
  [in] DWORD dwRowId,
  [out, retval] LONG* pcDeleted
);
```

pwszAuthority: See *pwszAuthority* definition in section 3.2.5.1.1.

dwFlags: An unsigned integer value that specifies the type of rows to be deleted. This parameter can be one of the following values:

Value	Meaning
0x00000000	Delete individual row.
0x00000001	Delete rows containing expired certificates.
0x00000002	Delete rows containing pending or failed requests.

FileTime: Contains a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). The value used to query for multiple rows to be deleted. MUST contain all zeros if dwRowId is nonzero.

dwTable: An unsigned integer value that specifies the table in which to delete rows. This parameter can be one of the following values:

Value	Meaning
0x00000000	Delete request table rows.
0x00003000	Delete extension table rows.
0x00004000	Delete attribute table rows.
0x00005000	Delete CRL table rows.

dwRowId: An unsigned integer value that represents the row identifier in the CA data table. MUST be set to 0 if FileTime is nonzero.

pcDeleted: Returned count of successfully deleted rows from the table.

The **DeleteRow** method is used to instruct the CA to delete rows from the specified table.

The following processing rules apply:

1. The CA MUST verify that exactly one of *dwRowId* or *FileTime* is zero. If both or neither are/is zero, the CA MUST fail the request.
2. The CA MUST verify that *dwTable* is set to one of the values defined the description for the *dwTable* parameter. If set to any other value, the CA MUST fail the request.
3. If dwTable is set to 0x00000000:
    1. If *dwFlags* is nonzero and not set to 0X00000001 or 0x00000002, the CA MUST fail the request.

2. If *dwRowId* is nonzero:

    1. If *dwRowId* is not a valid **RequestId**, the CA MUST pass the request and return 0 in the *pcDeleted* parameter.

    2. The CA MUST delete the corresponding request table rows, as well as all associated rows in the extension and attribute tables.

3. If FileTime is nonzero:

    - The CA MUST attempt to delete all request table rows that match the following criteria, as well as all associated rows in the extension and attribute tables.

        1. If dwFlags is set to 0x00000001:

            - All rows that contain issued and revoked certificates that expire before *FileTime*.

        2. If *dwFlags* is set to 0x00000002:

            - All rows that contain pending and failed requests that were last acted upon before *FileTime*.

4. If *dwTable* is set to 0x00003000:

    1. If *dwRowId* is zero, the CA MUST fail the request.

    2. If *dwFlags* is nonzero, the CA MUST fail the request.

    3. The CA MUST delete the corresponding extensions table row.

5. If dwTable is set to 0x00004000:

    1. If *dwRowId* is zero, the CA MUST fail the request.

    2. If *dwFlags* is nonzero, the CA MUST fail the request.

    3. The CA MUST delete the corresponding attributes table row.

6. If *dwTable* is set to 0x00005000:

    1. If *dwFlags* is nonzero and not set to 0x00000001, the CA MUST fail the request.

    2. If *dwFlags* is set to 0x00000000 or 0x00000001:

        1. If *dwRowId* is nonzero:

            1. If *dwRowId* is not a valid CRL table CRL.RowId, the CA MUST pass the request and return 0 in the *pcDeleted* parameter.

            2. The CA MUST delete the corresponding CRL table row.

        2. If *FileTime* is nonzero:

            - The CA MUST attempt to delete all CRL table rows that contain CRLs for which the value in the CRLNextUpdate column occurs before *FileTime*.

7. The CA MUST count all deleted rows and return that count in *pcDeleted.<64>

3.2.6 Timer Events

No timers.

3.2.7 Other Local Events

No other local events.

3.3 Algorithms

3.3.1 Sanitizing Common Names

The common name (CN) of the Active Directory (AD) objects, as specified in [MS-ADTS], that are used by the enrollment protocol are created by sanitizing the names of other objects and shortening the sanitized name so that it does not exceed 64 characters including spaces. The sanitized name MUST not exceed 64 characters in length. A name is sanitized by replacing disallowed characters with an exclamation point character (!) followed by four hexadecimal values that represent the 16-bit character being replaced.

The following rules apply to creating a sanitized common name (short name):

All disallowed characters in the original name MUST be replaced with the appropriate replacements values as specified in section 3.3.1.2.

The sanitized name must be truncated to no more than 51 characters in total length. The truncated name MUST NOT exceed 51 characters. If an incomplete sanitized character sequence remains at the end of the string (for example !002 instead of !0023), the incomplete sequence MUST be truncated completely.

The characters that were removed or truncated from the sanitized string in step 2 MUST be hashed according to the rules as specified in section 3.3.1.1. The resultant hash must be converted to a five-character string. The string MUST be 5 characters in total length, and MUST be padded with leading zeros on the left to ensure a total length of five characters.

A minus sign (–) should be appended to the truncated sanitized name followed by the five-character string that contains the hash value.

3.3.1.1 Hashing Processing Rules

The hash to represent truncated characters is computed by rotating a 16-bit value one bit to the left and adding each character truncated from the full common name (original name) until all of the truncated characters have been exhausted, as shown in the following example hash process rule:

If the string length of the full common name is less than 52 characters in total length, then the sanitized short name is the same as the full common name. Otherwise, the string base equals the first 51 characters of the full common name. The string excess equals characters 52 through the end of full common name.

For each character that is in excess of 51, the following algorithm will be applied to hash the excess characters:

- Hash is initialized with 0.
- For each excess character, the following calculation is performed:

- An unsigned 16-bit integer (LowBit) is calculated by using the following formula: ((0x8000 & Hash)? 1 : 0).
- The value of the Hash is recalculated by using the following formula: ((Hash << 1) | LowBit) + [excess character].

Next, the resultant hash equals the decimal representation of the calculated hash. The hash is left padded with zeros ('0') to ensure it is 5 characters in total length. The final short sanitized name equals the concatenation of the string base plus a '-' plus the five character hash.

3.3.1.2 Disallowed Characters

The following characters are disallowed and MUST NOT be used. The disallowed characters and their appropriate replacement values are noted below.

Name	Character	Value in !xxxx format
Exclamation mark	!	!0021
Inch or quotation mark	"	!0022
Number sign	#	!0023
Percent sign	%	!0025
Ampersand	&	!0026
Apostrophe	'	!0027
Opening parenthesis	(	!0028
Closing parenthesis	)	!0029
Asterisk	*	!002a
Plus sign	+	!002b
Comma	,	!002c
Slash	/	!002f
Colon	:	!003a
Semicolon	;	!003b
Less-than sign	<	!003c
Equals sign	=	!003d
Greater-than sign	>	!003e
Question mark	?	!003f
Left square bracket	[	!005b
Backslash	\	!005c
Right square bracket	]	!005d

Name	Character	Value in !xxxx format
Caret or circumflex accent	^	!005e
Grave accent	`	!0060
Left curly bracket	{	!007b
Pipe or vertical line	|	!007c
Right curly bracket	}	!007d

Characters whose value is less than 0x20 MUST be replaced with !00xx where xx is the hexadecimal value of the Unicode character. For example, the value of 0x10 will be replaced with !0010.Characters whose value is greater than or equal to 0x7F MUST be replaced with !00xx where xx is the hexadecimal value of the character. For example, the value of 0x80 will be replaced with !0080.

4 Protocol Examples

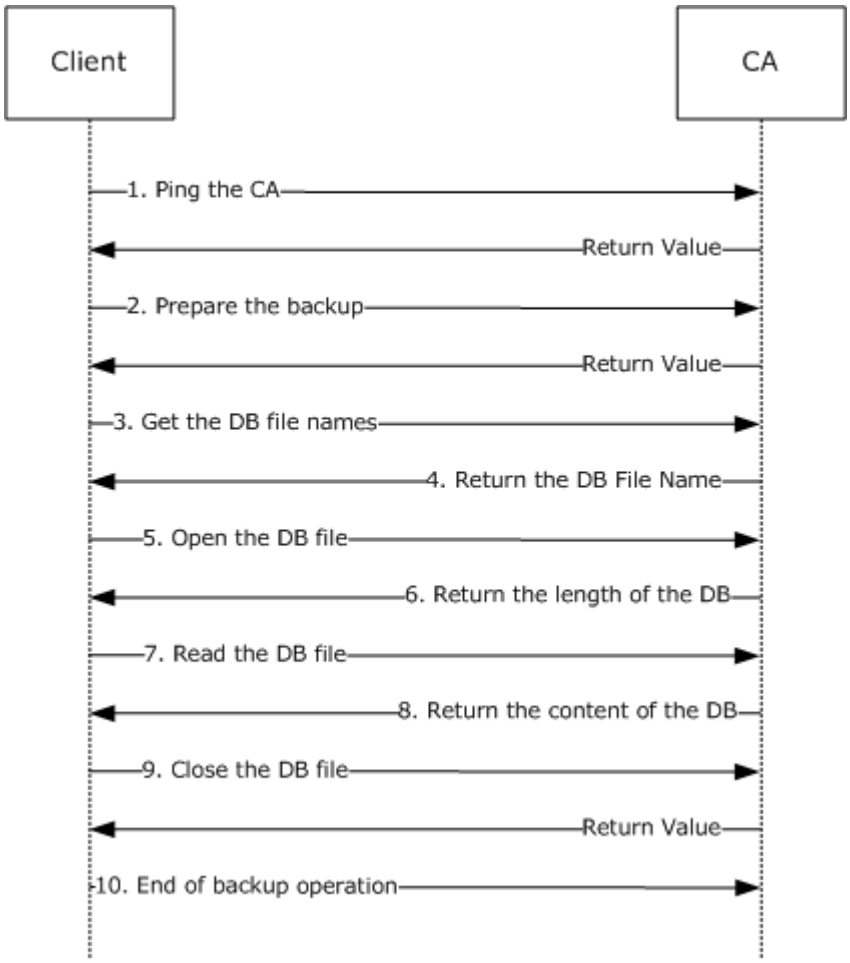


Figure 2: Backing up a CA database

A client determines that it is necessary to backup a CA database. To do so, the client needs to obtain the name of the CA to be backed up.

For this example, the CA name is "**RootCA**".

The following sequence of events needs to occur:

1. The client needs to ping the CA and validate that the CA service is online. The client calls the method Ping with the following parameters:

```
ICertAdminD::Ping(L"RootCA")
```

   The client verifies the return value from this method is S_OK (0).

2. The client needs to inform the CA that it would like to start a backup process. The client calls the BackupPrepare method with the following parameters:

```
ICertAdminD::BackupPrepare( L"RootCA", 0, 0, NULL, 0 )
```

The client verifies that the return value from this method is S_OK (0).

3. The client needs to retrieve the database file names. The client calls the BackupGetAttachmentInformation method with the following parameters:

```
ICertAdminD::BackupGetAttachementInformation
    (&pwszFileList,&cwList)
```

4. The server returns the number of database and log files by setting the out parameters cwList (in this example, it will be set to 1) and pwszFileList (in this example, it will be set to L"D\\servername\e$\winnt\system32\certlog\rootca.edb").

5. The client opens the specific database file returned in the previous step. The client would need to know the length of the file. The client calls the BackupOpenFile method with the following parameters:

```
ICertAdminD::BackupOpenFile
    (L"\\servername\e$\winnt\system32\certlog\rootca.edb",&Length)
```

6. The server returns the length of the requested file in the *Length* parameter. In this example, the length is 123456.

7. The client allocated the required buffer and reads the content of the DB file. The client calls the BackupReadFile method with the following parameters:

```
ICertAdminD::BackupReadFile(pbData,123456,&Read)
```

8. The server copies the content of the requested file to the *pbData* parameter and sets the length of the actual read operation into the *Read* parameter.

9. The client closes the DB file. The client calls BackupCloseFile with no parameters. The client verifies that the return value from this method is S_OK (0).

10. The client ends the backup operation by calling the BackupEnd method with no parameters.

5 Security

The protocol documented here allows an administrator to manipulate the CA in various ways.

The two that are most security-sensitive are that an administrator can:

- Manually approve **issuance** of a certificate.
- Recover an archived private key.

The CA has its own security requirements for keeping information from being tampered with and keeping cryptographic keys secret, as specified in [MS-WCCE] section 5. All of those requirements apply to an implementation of the CA. In addition, this protocol exposes a risk if the administrator is not authenticated properly and this section lists only additional requirements around that function.

5.1 Strong Administrator Authentication

An administrator of the CA must authenticate strongly. This could be via a high entropy password or some multiple-factor authentication method (such as smartcard). The CA administrator should use a login account that functions only for CA administration and not for any other function because use of the same credentials on a vulnerable computer while performing doing some other task exposes them to capture and misuse.

5.2 KDC Security

Because authentication of the administrator is by Kerberos, in this protocol, the KDC must itself be kept secure—free from tampering and free from vulnerabilities that would allow privilege-elevation penetrations.

5.3 Administrator Console Security

The administrator's console (the applications used by the administrator to run the client side of this protocol and the operating system in which that functionality runs) must be kept secure from penetration that would allow an attacker to act as the administrator.

5.4 Administrator Credential Issuance

Because the CA administrator is identified as some name in a Kerberos domain, for the purpose of access control by the CA, the human procedures for assigning a name to the CA administrator, adding that name to some named group of administrators and adding that group name to the **ACL** used by the CA, must be kept free from either penetration (for example, **social engineering**) or human mistake via common misspelling or unwarranted assumptions.

6 Appendix A: Full IDL

For ease of implementation, the full IDL is provided, where "ms-dtyp.idl" refers to the IDL found in [MS-DTYP] Appendix A, and "ms-oaut.idl" is the IDL found in [MS-OAUT] Appendix A.

```
import "ms-dtyp.idl";
import "ms-oaut.idl";

typedef byte BYTE;

typedef struct _CERTTRANSBLOB
{
    ULONG               cb;
    [size_is(cb), unique] BYTE *pb;
} CERTTRANSBLOB;

typedef struct _CATRANSPROP {
  LONG lPropID;
  BYTE propType;
  BYTE Reserved;
  USHORT propFlags;
  ULONG obwszDisplayName;
} CATRANSPROP;


typedef enum _ENUM_CATYPES
{
  ENUM_ENTERPRISE_ROOTCA = 0x00000000,
  ENUM_ENTERPRISE_SUBCA = 0x00000001,
  ENUM_STANDALONE_ROOTCA = 0x00000003,
  ENUM_STANDALONE_SUBCA = 0x00000004
} ENUM_CATYPES;

typedef struct _CAINFO{
  ULONG cbSize;
  ENUM_CATYPES CAType;
  ULONG cCASignatureCerts;
  ULONG cCAExchangeCerts;
  ULONG cExitModules;
  LONG lPropIDMax;
  LONG lRoleSeparationEnabled;
  ULONG cKRACertUsedCount;
  ULONG cKRACertCount;
  ULONG fAdvancedServer;
} CAINFO;




typedef struct _CERTTRANSDBATTRIBUTE
{
    ULONG obwszName;
    ULONG obwszValue;
} CERTTRANSDBATTRIBUTE;

typedef struct _CERTTRANSDBEXTENSION
{
```

```
    ULONG obwszName;
    LONG  ExtFlags;
    DWORD cbValue;
    ULONG obValue;
} CERTTRANSDBEXTENSION;


typedef struct _CERTTRANSDBCOLUMN
{
    DWORD Type;
    DWORD Index;
    DWORD cbMax;
    ULONG obwszName;
    ULONG obwszDisplayName;
} CERTTRANSDBCOLUMN;

typedef struct _CERTTRANSDBRESULTCOLUMN
{
    DWORD  Type;
    DWORD  Index;
    ULONG  obValue;
    DWORD  cbValue;
} CERTTRANSDBRESULTCOLUMN;

typedef struct _CERTTRANSDBRESULTROW// Marshaled form
{
    DWORD rowid;
    DWORD ccol;
    ULONG cbrow;
} CERTTRANSDBRESULTROW;


typedef struct _CERTVIEWRESTRICTION
{
    DWORD                           ColumnIndex;
    LONG                            SeekOperator;
    LONG                            SortOrder;
    [size_is(cbValue), unique] BYTE *pbValue;
    DWORD                           cbValue;
} CERTVIEWRESTRICTION;

/* Interface ICertAdminD */
[
    object,
    uuid(d99e6e71-fc88-11d0-b498-00a0c90312f3),
    helpstring("ICertAdmin DCOM Interface"),
    pointer_default(unique)
]
interface ICertAdminD: IUnknown
{
    HRESULT SetExtension(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 DWORD          dwRequestId,
        [in, string, unique] wchar_t const *pwszExtensionName,
        [in]                 DWORD          dwType,
        [in]                 DWORD          dwFlags,
        [in, ref]            CERTTRANSBLOB *pctbValue
    );
```

```
HRESULT SetAttributes(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          dwRequestId,
    [in, string, unique] wchar_t const *pwszAttributes
);

HRESULT ResubmitRequest(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          dwRequestId,
    [out]                DWORD         *pdwDisposition
);

HRESULT DenyRequest(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          dwRequestId
);

HRESULT IsValidCertificate(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in, string, unique] wchar_t const *pSerialNumber,
    [out]                LONG          *pRevocationReason,
    [out]                LONG          *pDisposition
);

HRESULT PublishCRL(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 FILETIME       FileTime
);

HRESULT GetCRL(
    [in, string, unique] wchar_t const *pwszAuthority,
    [out, ref]           CERTTRANSBLOB *pctbCRL
);

HRESULT RevokeCertificate(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in, string, unique] wchar_t const *pwszSerialNumber,
    [in]                 DWORD          Reason,
    [in]                 FILETIME       FileTime
);

HRESULT EnumViewColumn(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          iColumn,
    [in]                 DWORD          cColumn,
    [out]                DWORD         *pcColumn,
    [out, ref]           CERTTRANSBLOB *pctbColumnInfo
);

HRESULT GetViewDefaultColumnSet(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          iColumnSetDefault,
    [out]                DWORD         *pcColumn,
    [out, ref]           CERTTRANSBLOB *pctbColumnInfo
);

HRESULT EnumAttributesOrExtensions(
```

```
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          RowId,
    [in]                 DWORD          Flags,
    [in, string, unique] wchar_t const *pwszLast,
    [in]                 DWORD          celt,
    [out]                DWORD         *pceltFetched,
    [out, ref]           CERTTRANSBLOB *pctbOut
);


HRESULT OpenView(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                   DWORD                    ccvr,
    [in, size_is(ccvr)]    CERTVIEWRESTRICTION const *acvr,
    [in]                   DWORD                     ccolOut,
    [in, size_is(ccolOut)] DWORD const              *acolOut,
    [in]                   DWORD                        ielt,
    [in]                   DWORD                        celt,
    [out]                  DWORD               *pceltFetched,
    [out, ref]             CERTTRANSBLOB      *pctbResultRows

);


HRESULT EnumView(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          ielt,
    [in]                 DWORD          celt,
    [out]                DWORD         *pceltFetched,
    [out, ref]           CERTTRANSBLOB *pctbResultRows
);


HRESULT CloseView(
    [in, string, unique] wchar_t const *pwszAuthority
);

HRESULT ServerControl(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 DWORD          dwControlFlags,
    [out, ref]           CERTTRANSBLOB *pctbOut
);

/* this is a test function */
HRESULT Ping(
    [in, string, unique] wchar_t const *pwszAuthority
);

HRESULT GetServerState(
    [in, string, unique] wchar_t const *pwszAuthority,
    [out]                DWORD         *pdwState
);

HRESULT BackupPrepare(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                 unsigned long  grbitJet,
    [in]                 unsigned long  dwBackupFlags,
    [in]                 WCHAR const   *pwszBackupAnnotation,
```

```
        [in]                DWORD          dwClientIdentifier
    );

    HRESULT BackupEnd(
    );

    HRESULT BackupGetAttachmentInformation(
        [out, size_is( , *pcwcDBFiles)] WCHAR **ppwszzDBFiles,
        [out]                           LONG   *pcwcDBFiles
    );

    HRESULT BackupGetBackupLogs(
        [out, size_is( , *pcwcLogFiles)] WCHAR **ppwszzLogFiles,
        [out]                            LONG   *pcwcLogFiles
    );

    HRESULT BackupOpenFile(
        [in, string, unique] wchar_t const  *pwszPath,
        [out]                unsigned hyper *pliLength
    );

    HRESULT BackupReadFile(
        [ref, out, size_is(cbBuffer)] BYTE *pbBuffer,
        [in]                          LONG  cbBuffer,
        [out]                         LONG *pcbRead
    );

    HRESULT BackupCloseFile(
    );

    HRESULT BackupTruncateLogs(
    );

    HRESULT ImportCertificate(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in, ref]               CERTTRANSBLOB *pctbCertificate,
        [in]                    LONG           dwFlags,
        [out]                   LONG          *pdwRequestId
    );

    HRESULT BackupGetDynamicFiles(
        [out, size_is( , *pcwcFiles)] WCHAR **ppwszzFiles,
        [out]                         LONG   *pcwcFiles
    );

    HRESULT RestoreGetDatabaseLocations(
        [out, size_is( , *pcwcPaths)] WCHAR
                      **ppwszzDatabaseLocations,
        [out]                         LONG   *pcwcPaths
    );
};


/* Interface ICertAdminD2 */
[
    object,
    uuid(7fe0d935-dda6-443f-85d0-1cfb58fe41dd),
    helpstring("ICertAdmin2 DCOM Interface"),
```

```
    pointer_default(unique)
]
interface ICertAdminD2: ICertAdminD
{
    HRESULT PublishCRLs(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 FILETIME       FileTime,
        [in]                 DWORD            Flags
    );

    HRESULT GetCAProperty(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 LONG           PropId,
        [in]                 LONG           PropIndex,
        [in]                 LONG           PropType,
        [out, ref]           CERTTRANSBLOB *pctbPropertyValue
    );

    HRESULT SetCAProperty(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 LONG           PropId,
        [in]                 LONG           PropIndex,
        [in]                 LONG           PropType,
        [in]                 CERTTRANSBLOB *pctbPropertyValue
    );

    HRESULT GetCAPropertyInfo(
        [in, string, unique] wchar_t const *pwszAuthority,
        [out]                LONG          *pcProperty,
        [out, ref]           CERTTRANSBLOB *pctbPropInfo
    );

    HRESULT EnumViewColumnTable(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in]                 DWORD          iTable,
        [in]                 DWORD          iColumn,
        [in]                 DWORD          cColumn,
        [out]                DWORD         *pcColumn,
        [out, ref]           CERTTRANSBLOB *pctbColumnInfo
    );

    HRESULT GetCASecurity(
        [in, string, unique] wchar_t const *pwszAuthority,
        [out, ref]             CERTTRANSBLOB *pctbSD
    );

    HRESULT SetCASecurity(
        [in, string, unique] wchar_t const *pwszAuthority,
        [in, ref]             CERTTRANSBLOB *pctbSD
    );

    /* this is a test function */
    HRESULT Ping2(
        [in, string, unique] wchar_t const *pwszAuthority
    );

    HRESULT GetArchivedKey(
        [in, string, unique] wchar_t const *pwszAuthority,
```

```
    [in]                    DWORD            dwRequestId,
    [out, ref]            CERTTRANSBLOB *pctbArchivedKey
);


HRESULT GetAuditFilter(
    [in, string, unique] wchar_t const *pwszAuthority,
    [out]               DWORD         *pdwFilter
);

HRESULT SetAuditFilter(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                DWORD          dwFilter
);

HRESULT GetOfficerRights(
    [in, string, unique] wchar_t const *pwszAuthority,
    [out]                    BOOL           *pfEnabled,
    [out, ref]            CERTTRANSBLOB *pctbSD
);

HRESULT SetOfficerRights(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                    BOOL            fEnable,
    [in, ref]             CERTTRANSBLOB *pctbSD
);

HRESULT GetConfigEntry(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in, string, unique] wchar_t const *pwszNodePath,
    [in, string, ref]    wchar_t const *pwszEntry,
    [out, ref]             VARIANT           *pVariant
);

HRESULT SetConfigEntry(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in, string, unique] wchar_t const *pwszNodePath,
    [in, string, ref]    wchar_t const *pwszEntry,
    [in, ref]             VARIANT           *pVariant
);

HRESULT ImportKey(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                    DWORD            dwRequestId,
    [in, string, unique] wchar_t const *pwszCertHash,
    [in]                    DWORD            dwFlags,
    [in, ref]             CERTTRANSBLOB *pctbKey
);

HRESULT GetMyRoles(
    [in, string, unique] wchar_t const *pwszAuthority,
    [out]               LONG          *pdwRoles
);

HRESULT DeleteRow(
    [in, string, unique] wchar_t const *pwszAuthority,
    [in]                    DWORD            dwFlags,
    [in]                FILETIME       FileTime,
```

```
        [in]                DWORD          dwTable,
        [in]                DWORD          dwRowId,
        [out, retval]       LONG          *pcDeleted
    );
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Server 2003
- Windows Vista
- Windows XP
- Windows 2000

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription

<1> Section 1.1: Windows Server 2003 stores request submissions and certificate revocation that has occurred since the last log file truncation or backup. Log file volume increases as database activity occurs. The log files can be decreased in size by performing a backup and then calling BackupTruncateLogs (as specified in section 2.2.2.1).

<2> Section 1.3: In Windows implementations, an Active Directory domain controller includes a Key Distribution Center (KDC) as specified in [MS-KILE].

<3> Section 1.3.1.5: certificate templates are implemented as objects that reside in the CN=Certificate Templates, CN=Public Key Services, CN=Services, CN=Configuration, DC=ForestRootDomain Active Directory **container**. They can be retrieved by any **domain** member using the Lightweight Directory Access Protocol, as specified in [MS-ADTS]. Information on Active Directory containers is also as specified in [MS-ADTS]. Information on the format of Microsoft certificate templates can be found in [MSFT-TEMPLATES].

<4> Section 1.5.2: Windows-based clients discover Windows CAs by reading the certificate enrollment object in the AD (as specified by [MS-ADTS]) and by using the Lightweight Directory Access Protocol (as specified in [RFC2559]).

The Enrollment Object that defines the names of the CAs is located under CN=Enrollment Services,CN=Public Key Services,CN=Services,CN=Configuration,DC=ForestRootDomain container of the Active Directory. Each CA will have an entry with a class of **pKIEnrollmentService**, as specified in [MS-ADSC] section 2.181.

The **cn** attribute of **pKIEnrollmentService** is the CA name. The **dNSHostName** attribute ([MS-ADA1] section 2.184) of **pKIEnrollmentService** contains the machine name that hosts the CA service.

<5> Section 1.7: Windows-based clients query for the certificate services interface version. If certificate services supports ICertAdminD2, it will be used; otherwise, ICertAdminD is used.

<6> Section 2.1: Windows-based clients and servers require encryption, authentication, and integrity protection from the RPC connection that is used for this protocol, as specified in [MS-DCOM] and [MS-RPCE]. If the connection is not authenticated, the CA will refuse to establish a connection with the client.

<7> Section 2.2.1.9: The Windows CA assigns a value of 2 if the extension was added by the **policy module** of the CA.

<8> Section 2.2.1.9: The Windows CA assigns a value of 3 if the extension was added interactively by a human administrator of the CA.

<9> Section 2.2.1.9: The Windows CA assigns a value of 4 if the extension was added by the certificate server engine and not the policy module component of the CA.

<10> Section 2.2.2.1: Windows Server 2003 uses CA exchange certificates that contain the following X.509v3 extensions specific to Windows:

- Application Policies (Policy Identifier = Private Key Archival)
- Certificate Template Name
- Certificate Template Information

A Windows Server 2003 CA automatically generates an exchange certificate for use of the key archival during certificate enrollment. By default, a Windows CA uses the CA Exchange Certificate template to construct the certificate for use by the CA. The CA Exchange Certificate template, described in [MSFT-TEMPLATES], defines the attributes, extensions, and validity period of the CA exchange certificate.

<11> Section 2.2.2.2:

Windows Server 2003 will use key recovery certificates that contain the following X.509v3 extensions specific to Windows:

- Application Policies (Policy Identifier = Key Recovery Agent)
- Certificate Template Information

key recovery certificates, when issued by a Windows enterprise CA, are automatically written to the configuration container of Active Directory. The actual certificates are published to the userCertificate attribute (as specified in [RFC4523]) of the key recovery agent (KRA) object when issued to a member of the domain administrators group in Active Directory.

<12> Section 3.1.4: Windows allows managing a CA through the use of Microsoft Management Console (MMC) and through the use of command-line tools (certutil.exe).

<13> Section 3.1.5.1.14: If the DCOM connection to a Microsoft Windows CA, on which a view exists, is terminated before a call to CloseView is made, Windows Server eventually releases the resources associated with the view, but might not do so immediately.

<14> Section 3.1.5.1.25: Microsoft management clients call the BackupTruncateLogs method after performing full backup operation.

<15> Section 3.2.1.1.1: Windows uses a **DWORD** number to represent these values. The following table corresponds Windows internal values to the string representations above:

Windows value	Abstract data model value
CR_DISP_ERROR 0x00000001	"request failed"
CR_DISP_DENIED 0x00000002	"request denied"
CR_DISP_ISSUED	"Certificate issued"

Windows value	Abstract data model value
0x00000003	
CR_DISP_UNDER_SUBMISSION 0x00000005	"request pending"
CR_DISP_REVOKED 0x00000006	"certificate revoked"

<16> Section 3.2.1.1.1: The Microsoft CA will not put a certificate's serial number in a base or delta CRL until the time specified in Request.Revocation.Date for that certificate has passed.

Request.Revocation.Date is implemented as Request.RevokedEffectiveWhen in the Microsoft Windows CA database request table.

<17> Section 3.2.1.4.2: The Microsoft CA indexes the Attribute table RequestID field, the Extension table RequestID field, and the CRL table CRL.RowId field.

<18> Section 3.2.1.4.2: Windows implements the version-specific Request, Extension, Attribute, and CRL database tables as detailed below. The following table details the Request table for Windows 2000 Server :

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x1000	0x10001	4 bytes	"Request.RequestID" (Request.Request.ID)	"Request ID"
0x1001	0x3	65536 bytes	"Request.RawRequest" (Request.Raw.Request)	"Binary Request"
0x1002	0x3	16384 bytes	"Request.RawOldCertificates"	"Old Certificate"
0x1003	0x4	32768 bytes	"Request.RequestAttributes"	"Request Attributes"
0x1004	0x1	4 bytes	"Request.RequestType"	"Request Type"
0x1005	0x1	4 bytes	"Request.RequestFlags"	"Request Flags"
0x1006	0x1	4 bytes	"Request.Status"	"Request Status"
0x1007	0x1	4 bytes	"Request.StatusCode"	"Request Status Code"
0x1008	0x10001	4 bytes	"Request.Disposition" (Request.Disposition)	"Request Disposition"
0x1009	0x4	4 bytes	"Request.DispositionMessage"	"Request Disposition Message"
0x100a	0x2	8 bytes	"Request.SubmittedWhen"	"Request Submission Date"
0x100b	0x2	8 bytes	"Request.ResolvedWhen"	"Request Resolution Date"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x100c	0x2	8 bytes	"Request.RevokedWhen"	"Revocation Date"
0x100d	0x2	8 bytes	"Request.RevokedEffectiveWhen" (Request.Revocation.Date)	"Effective Revocation Date"
0x100e	0x1	4 bytes	"Request.RevokedReason" (Request.Revoked.Reason)	"Revocation Reason"
0x100f	0x4	2048 bytes	"Request.RequesterName"	"Requester Name"
0x1010	0x4	2048 bytes	"Request.RequesterAddress"	"Requester Address"
0x1011	0x4	8192 bytes	"Request.DistinguishedName"	"Request Distinguished Name"
0x1012	0x3	4096 bytes	"Request.RawName"	"Request Binary Name"
0x1013	0x1	4 bytes	"Request.NameType"	"Request Name Type"
0x1014	0x4	8192 bytes	"Request.Country"	"Request Country/Region"
0x1015	0x4	8192 bytes	"Request.Organization"	"Request Organization"
0x1016	0x4	8192 bytes	"Request.OrgUnit"	"Request Organization Unit"
0x1017	0x4	8192 bytes	"Request.CommonName"	"Request Common Name"
0x1018	0x4	8192 bytes	"Request.Locality"	"Request City"
0x1019	0x4	8192 bytes	"Request.State"	"Request State"
0x101a	0x4	8192 bytes	"Request.Title"	"Request Title"
0x101b	0x4	8192 bytes	"Request.GivenName"	"Request First Name"
0x101c	0x4	8192 bytes	"Request.Initials"	"Request Initials"
0x101d	0x4	8192 bytes	"Request.SurName"	"Request Last Name"
0x101e	0x4	8192 bytes	"Request.DomainComponent"	"Request Domain Component"
0x101f	0x4	8192 bytes	"Request.EMail"	"Request Email Address"
0x1020	0x4	8192 bytes	"Request.StreetAddress"	"Request Street Address"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x1021	0x4	8192 bytes	"Request.UnstructuredName"	"Request Unstructured Name"
0x1022	0x4	8192 bytes	"Request.UnstructuredAddress"	"Request Unstructured Address"
0x1023	0x4	8192 bytes	"Request.DeviceSerialNumber"	"Request Device Serial Number"
0x2000	0x10001	4 bytes	"RequestID"	"Issued Request ID"
0x2001	0x3	16384 bytes	"RawCertificate" (Raw.Certificate)	"Binary Certificate"
0x2002	0x4	128 bytes	"CertificateHash"	"Certificate Hash"
0x2003	0x4	254 bytes	"CertificateType"	"Certificate Type"
0x2004	0x10004	128 bytes	"SerialNumber" (Serial.Number)	"Serial Number"
0x2005	0x1	4 bytes	"IssuerNameId"	"Issuer Name ID"
0x2006	0x2	8 bytes	"NotBefore"	"Certificate Effective Date"
0x2007	0x10002	8 bytes	"NotAfter"	"Certificate Expiration Date"
0x2008	0x3	4096 bytes	"RawPublicKey"	"Binary Public Key"
0x2009	0x4	254 bytes	"PublicKeyAlgorithm"	"Public Key Algorithm"
0x200a	0x3	4096 bytes	"RawPublicKeyAlgorithmParameters"	"Public Key Algorithm Parameters"
0x200b	0x4	8192 bytes	"DistinguishedName"	"Issued Distinguished Name"
0x200c	0x3	4096bytes	"RawName"	"Issued Binary Name"
0x200d	0x1	4 bytes	"NameType"	"Issued Name Type"
0x200e	0x4	8192 bytes	"Country"	"Issued Country/Region"
0x200f	0x4	8192 bytes	"Organization"	"Issued Organization"
0x2010	0x4	8192 bytes	"OrgUnit"	"Issued Organization Unit"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x2011	0x10004	8192 bytes	"CommonName"	"Issued Common Name"
0x2012	0x4	8192 bytes	"Locality"	"Issued City"
0x2013	0x4	8192 bytes	"State"	"Issued State"
0x2014	0x4	8192 bytes	"Title"	"Issued Title"
0x2015	0x4	8192 bytes	"GivenName"	"Issued First Name"
0x2016	0x4	8192 bytes	"Initials"	"Issued Initials"
0x2017	0x4	8192 bytes	"SurName"	"Issued Last Name"
0x2018	0x4	8192 bytes	"DomainComponent"	"Issued Domain Component"
0x2019	0x4	8192 bytes	"EMail"	"Issued Email address"
0x201a	0x4	8192 bytes	"StreetAddress"	"Issued Street Address"
0x201b	0x4	8192 bytes	"UnstructuredName"	"Issued Unstructured Name"
0x201c	0x4	8192 bytes	"UnstructuredAddress"	"Issued Unstructured Address"
0x201d	0x4	8192 bytes	"DeviceSerialNumber"	"Issued Device Serial Number"
0x201e	0x3	16384 bytes	"RawSMIMECapabilities"	"Issued SMIME Capabilities"

The following table details the Request Table for Windows Server 2003:

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
0x1000	0x10001	4 bytes	"Request.RequestID" (Request.Request.ID)	"Request ID"
0x1001	0x3	65536 bytes	"Request.RawRequest" (Request.Raw.Request)	"Binary Request"
0x1002	0x3	65536 bytes	"Request.RawArchivedKey" (Request.Raw.ArchivedKey)	"Archived Key"
0x1003	0x4	8192 bytes	"Request.KeyRecoveryHashes"	"Key Recovery Agent Hashes"

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
0x1004	0x3	16384 bytes	"Request.RawOldCertificates"	"Old Certificate"
0x1005	0x4	32768 bytes	"Request.RequestAttributes"	"Request Attributes"
0x1006	0x1	4 bytes	"Request.RequestType"	"Request Type"
0x1007	0x1	4 bytes	"Request.RequestFlags"	"Request Flags"
0x1008	0x1	4 bytes	"Request.StatusCode"	"Request Status Code"
0x1009	0x10001	4 bytes	"Request.Disposition" (Request.Disposition)	"Request Disposition"
0x100a	0x4	4 bytes	"Request.DispositionMessage"	"Request Disposition Message"
0x100b	0x10002	8 bytes	"Request.SubmittedWhen"	"Request Submission Date"
0x100c	0x10002	8 bytes	"Request.ResolvedWhen"	"Request Resolution Date"
0x100d	0x2	8 bytes	"Request.RevokedWhen"	"Revocation Date"
0x100e	0x10002	8 bytes	"Request.RevokedEffectiveWhen"(Reqeust.Revocation.Date)	"Effective Revocation Date"
0x100f	0x1	4 bytes	"Request.RevokedReason"(Request.Revoked.Reason)	"Revocation Reason"
0x1010	0x10004	2048 bytes	"Request.RequesterName"	"Requester Name"
0x1011	0x10004	2048 bytes	"Request.CallerName"	"Caller Name"
0x1012	0x4	8192 bytes	"Request.SignerPolicies"	"Signer Policies"
0x1013	0x4	8192 bytes	"Request.SignerApplicationPolicies"	"Signer Application Policies"
0x1014	0x1	4 bytes	"Request.Officer"	"Officer"
0x1015	0x4	8192	"Request.DistinguishedName"	"Request

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
		bytes		Distinguished Name"
0x1016	0x3	4096 bytes	"Request.RawName"	"Request Binary Name"
0x1017	0x4	8192 bytes	"Request.Country"	"Request Country/Region"
0x1018	0x4	8192 bytes	"Request.Organization"	"Request Organization"
0x1019	0x4	8192 bytes	"Request.OrgUnit"	"Request Organization Unit"
0x101a	0x4	8192 bytes	"Request.CommonName"	"Request Common Name"
0x101b	0x4	8192 bytes	"Request.Locality"	"Request City"
0x101c	0x4	8192 bytes	"Request.State"	"Request State"
0x101d	0x4	8192 bytes	"Request.Title"	"Request Title"
0x101e	0x4	8192 bytes	"Request.GivenName"	"Request First Name"
0x101f	0x4	8192 bytes	"Request.Initials"	"Request Initials"
0x1020	0x4	8192 bytes	"Request.SurName"	"Request Last Name"
0x1021	0x4	8192 bytes	"Request.DomainComponent"	"Request Domain Component"
0x1022	0x4	8192 bytes	"Request.EMail"	"Request Email Address"
0x1023	0x4	8192 bytes	"Request.StreetAddress"	"Request Street Address"
0x1024	0x4	8192 bytes	"Request.UnstructuredName"	"Request Unstructured Name"
0x1025	0x4	8192 bytes	"Request.UnstructuredAddress"	"Request Unstructured

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
				Address"
0x1026	0x4	8192 bytes	"Request.DeviceSerialNumber"	"Request Device Serial Number"
0x2000	0x10001	4 bytes	"RequestID"	"Issued Request ID"
0x2001	0x3	16384 bytes	"RawCertificate" (Raw.Certificate)	"Binary Certificate"
0x2002	0x10004	128 bytes	"CertificateHash"	"Certificate Hash"
0x2003	0x10004	254 bytes	"CertificateTemplate"	"Certificate Template"
0x2004	0x1	4 bytes	"EnrollmentFlags"	"Template Enrollment Flags"
0x2005	0x1	4 bytes	"GeneralFlags"	"Template General Flags"
0x2006	0x10004	128 bytes	"SerialNumber" (Serial.Number)	"Serial Number"
0x2007	0x1	4 bytes	"IssuerNameId"	"Issuer Name ID"
0x2008	0x2	8 bytes	"NotBefore"	"Certificate Effective Date"
0x2009	0x10002	8 bytes	"NotAfter"	"Certificate Expiration Date"
0x200a	0x4	128 bytes	"SubjectKeyIdentifier"	"Issued Subject Key Identifier"
0x200b	0x3	4096 bytes	"RawPublicKey"	"Binary Public Key"
0x200c	0x1	4 bytes	"PublicKeyLength"	"Public Key Length"
0x200d	0x4	254 bytes	"PublicKeyAlgorithm"	"Public Key Algorithm"
0x200e	0x3	4096 bytes	"RawPublicKeyAlgorithmParameters"	"Public Key Algorithm Parameters"
0x200f	0x1000	2048	"UPN"	"User Principal

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
	4	bytes		Name"
0x2010	0x4	8192 bytes	"DistinguishedName"	"Issued Distinguished Name"
0x2011	0x3	4096bytes	"RawName"	"Issued Binary Name"
0x2012	0x4	8192 bytes	"Country"	"Issued Country/Region"
0x2013	0x4	8192 bytes	"Organization"	"Issued Organization"
0x2014	0x4	8192 bytes	"OrgUnit"	"Issued Organization Unit"
0x2015	0x10004	8192 bytes	"CommonName"	"Issued Common Name"
0x2016	0x4	8192 bytes	"Locality"	"Issued City"
0x2017	0x4	8192 bytes	"State"	"Issued State"
0x2018	0x4	8192 bytes	"Title"	"Issued Title"
0x2019	0x4	8192 bytes	"GivenName"	"Issued First Name"
0x201a	0x4	8192 bytes	"Initials"	"Issued Initials"
0x201b	0x4	8192 bytes	"SurName"	"Issued Last Name"
0x201c	0x4	8192 bytes	"DomainComponent"	"Issued Domain Component"
0x201d	0x4	8192 bytes	"EMail"	"Issued Email address"
0x201e	0x4	8192 bytes	"StreetAddress"	"Issued Street Address"
0x201f	0x4	8192 bytes	"UnstructuredName"	"Issued Unstructured Name"

Column index	Data Type	Maximum size of data	Column name (ADM Element)	Column display name
0x2020	0x4	8192 bytes	"UnstructuredAddress"	"Issued Unstructured Address"
0x2021	0x4	8192 bytes	"DeviceSerialNumber"	"Issued Device Serial Number"

The following table details the Request table for Windows Server 2008:

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x1000	0x10001	4 bytes	Request.RequestID (Request.Request.ID)	"Request ID"
0x1001	0x3	65536 bytes	"Request.RawRequest" (Request.Raw.Request)	"Binary Request"
0x1002	0x3	65536 bytes	"Request.RawArchivedKey"(Request.Raw.Archived.Key)	"Archived Key"
0x1003	0x4	8192 bytes	"Request.KeyRecoveryHashes"	"Key Recovery Agent Hashes"
0x1004	0x3	16384 bytes	"Request.RawOldCertificates"	"Old Certificate"
0x1005	0x4	32768 bytes	"Request.RequestAttributes"	"Request Attributes"
0x1006	0x1	4 bytes	"Request.RequestType"	"Request Type"
0x1007	0x1	4 bytes	"Request.RequestFlags"	"Request Flags"
0x1008	0x1	4 bytes	"Request.StatusCode"	"Request Status Code"
0x1009	0x10001	4 bytes	"Request.Disposition" (Request.Disposition)	"Request Disposition"
0x100a	0x4	4 bytes	"Request.DispositionMessage"	"Request Disposition Message"
0x100b	0x10002	8 bytes	"Request.SubmittedWhen"	"Request Submission Date"
0x100c	0x10002	8 bytes	"Request.ResolvedWhen"	"Request Resolution Date"
0x100d	0x2	8 bytes	"Request.RevokedWhen"	"Revocation Date"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x100e	0x10002	8 bytes	"Request.RevokedEffectiveWhen" (Request.Revocation.Date)	"Effective Revocation Date"
0x100f	0x1	4 bytes	"Request.RevokedReason" (Request.Revoked.Reason)	"Revocation Reason"
0x1010	0x10004	2048 bytes	"Request.RequesterName"	"Requester Name"
0x1011	0x10004	2048 bytes	"Request.CallerName"	"Caller Name"
0x1012	0x4	8192 bytes	"Request.SignerPolicies"	"Signer Policies"
0x1013	0x4	8192 bytes	"Request.SignerApplicationPolicies"	"Signer Application Policies"
0x1014	0x1	4 bytes	"Request.Officer"	"Officer"
0x1015	0x4	8192 bytes	"Request.DistinguishedName"	"Request Distinguished Name"
0x1016	0x3	4096 bytes	"Request.RawName"	"Request Binary Name"
0x1017	0x4	8192 bytes	"Request.Country"	"Request Country/Region"
0x1018	0x4	8192 bytes	"Request.Organization"	"Request Organization"
0x1019	0x4	8192 bytes	"Request.OrgUnit"	"Request Organization Unit"
0x101a	0x4	8192 bytes	"Request.CommonName"	"Request Common Name"
0x101b	0x4	8192 bytes	"Request.Locality"	"Request City"
0x101c	0x4	8192 bytes	"Request.State"	"Request State"
0x101d	0x4	8192 bytes	"Request.Title"	"Request Title"
0x101e	0x4	8192 bytes	"Request.GivenName"	"Request First Name"
0x101f	0x4	8192 bytes	"Request.Initials"	"Request Initials"
0x1020	0x4	8192 bytes	"Request.SurName"	"Request Last Name"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x1021	0x4	8192 bytes	"Request.DomainComponent"	"Request Domain Component"
0x1022	0x4	8192 bytes	"Request.EMail"	"Request Email Address"
0x1023	0x4	8192 bytes	"Request.StreetAddress"	"Request Street Address"
0x1024	0x4	8192 bytes	"Request.UnstructuredName"	"Request Unstructured Name"
0x1025	0x4	8192 bytes	"Request.UnstructuredAddress"	"Request Unstructured Address"
0x1026	0x4	8192 bytes	"Request.DeviceSerialNumber"	"Request Device Serial Number"
0x2000	0x10001	4 bytes	"RequestID"	"Issued Request ID"
0x2001	0x3	16384 bytes	"RawCertificate" (Raw.Certificate)	"Binary Certificate"
0x2002	0x10004	128 bytes	"CertificateHash"	"Certificate Hash"
0x2003	0x10004	254 bytes	"CertificateTemplate"	"Certificate Template"
0x2004	0x1	4 bytes	"EnrollmentFlags"	"Template Enrollment Flags"
0x2005	0x1	4 bytes	"GeneralFlags"	"Template General Flags"
0x2006	0x10004	128 bytes	"SerialNumber" (Serial.Number)	"Serial Number"
0x2007	0x1	4 bytes	"IssuerNameId"	"Issuer Name ID"
0x2008	0x2	8 bytes	"NotBefore"	"Certificate Effective Date"
0x2009	0x10002	8 bytes	"NotAfter"	"Certificate Expiration Date"
0x200a	0x4	128 bytes	"SubjectKeyIdentifier"	"Issued Subject Key Identifier"
0x200b	0x3	4096 bytes	"RawPublicKey"	"Binary Public Key"
0x200c	0x1	4 bytes	"PublicKeyLength"	"Public Key Length"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x200d	0x4	254 bytes	"PublicKeyAlgorithm"	"Public Key Algorithm"
0x200e	0x3	4096 bytes	"RawPublicKeyAlgorithmParameters"	"Public Key Algorithm Parameters"
0x200f	0x1	4 bytes	"PublishExpiredCertInCRL" (Publish.Expired.Cert.In.CRL)	"PublishExpiredCertInCRL"
0x2010	0x10004	2048 bytes	"UPN"	"User Principal Name"
0x2011	0x4	8192 bytes	"DistinguishedName"	"Issued Distinguished Name"
0x2012	0x3	4096bytes	"RawName"	"Issued Binary Name"
0x2013	0x4	8192 bytes	"Country"	"Issued Country/Region"
0x2014	0x4	8192 bytes	"Organization"	"Issued Organization"
0x2015	0x4	8192 bytes	"OrgUnit"	"Issued Organization Unit"
0x2016	0x10004	8192 bytes	"CommonName"	"Issued Common Name"
0x2017	0x4	8192 bytes	"Locality"	"Issued City"
0x2018	0x4	8192 bytes	"State"	"Issued State"
0x2019	0x4	8192 bytes	"Title"	"Issued Title"
0x201a	0x4	8192 bytes	"GivenName"	"Issued First Name"
0x201b	0x4	8192 bytes	"Initials"	"Issued Initials"
0x201c	0x4	8192 bytes	"SurName"	"Issued Last Name"
0x201d	0x4	8192 bytes	"DomainComponent"	"Issued Domain Component"
0x201e	0x4	8192 bytes	"EMail"	"Issued Email address"
0x201f	0x4	8192 bytes	"StreetAddress"	"Issued Street Address"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x2020	0x4	8192 bytes	"UnstructuredName"	"Issued Unstructured Name"
0x2021	0x4	8192 bytes	"UnstructuredAddress"	"Issued Unstructured Address"
0x2022	0x4	8192 bytes	"DeviceSerialNumber"	"Issued Device Serial Number"

The following table details the Extension table for Windows 2000 Server/Windows Server 2003/Windows Server 2008:

Column index	Data type	Maximum size of data	Column name	Column display name
0x4000	0x1001	4 bytes	"ExtensionRequestId"	"Extension Request Id"
0x4001	0x4	254 bytes	"ExtensionName"	"Extension Name"
0x4002	0x1	4 bytes	"ExtensionFlags"	"Extension Flags"
0x4003	0x3	4096 bytes	"ExtensionRawValue"	"Extension Raw Value"

The following table details the Attribute table for Windows 2000 Server/Windows Server 2003/Windows Server 2008:

Column index	Data type	Maximum size of data	Column name	Column display name
0x3000	0x10001	4 bytes	"AttributeRequestId")	"Attribute Request Id"
0x3001	0x4	254 bytes	"AttributeName"	"Attribute Name"
0x3002	0x4	8192 bytes	"AttributeValue"	"Attribute Value"

The following table details the CRL table for Windows Server 2003 and Windows Server 2008 (Windows 2000 Server does not have a CRL table).

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x5000	0x10001	4 bytes	CRLRowId (CRL.RowID)	"CRL Row ID"
0x5001	0x10001	4 bytes	"CRLNumber"	"CRL Number"
0x5002	0x1	4 bytes	"CRLMinBase"	"CRL Minimum Base"
0x5003	0x1	4 bytes	"CRLNameId"	"CRL Name ID"
0x5004	0x1	4 bytes	"CRLCount"	"CRL Count"
0x5005	0x2	8 bytes	"CRLThisUpdate"	"CRL This Update"

Column index	Data type	Maximum size of data	Column name (ADM Element)	Column display name
0x5006	0x10002	8 bytes	"CRLNextUpdate"	"CRL Next Update"
0x5007	0x2	8 bytes	"CRLThisPublish"	"CRL This Publish"
0x5008	0x10002	8 bytes	"CRLNextPublish"	"CRL Next Publish"
0x5009	0x2	8 bytes	"CRLEffective"	"CRL Effective"
0x500a	0x10002	8 bytes	"CRLPropagationComplete"	"CRL Propagation Complete"
0x500b	0x10002	8 bytes	"CRLLastPublish"	"CRL Last Published"
0x500c	0x10001	4 bytes	"CRLPublishAttempts"	"CRL Publish Attempts"
0x500d	0x1	4 bytes	"CRLPublishFlags"	"CRL Publish Flags"
0x500e	0x10001	4 bytes	"CRLPublishStatusCode" (CRL.Publish.Status.Code)	"CRL Publish Status Code"
0x500f	0x4	8192 bytes	"CRLPublishError"	"CRL Publish Error Information"
0x5010	0x3	536870912 bytes	"CRLRawCRL" (CRL.Raw.CRL)	"CRL Raw CRL"

<19> Section 3.2.1.7: A Windows CA defines six permissions: Enroll, Read, Officer, Administrator, Operator, and Auditor.

For CA security (GetCASecurity, SetCASecurity, and GetMyRoles), the Microsoft Certificate Authority assigns permissions to principals (identified by the access control entry (ACE)) in the following manner:

Permission	Bit value
Read	0x00000100
Enroll	0x00000200
Officer	0x00000002
Administrator	0x00000001
Auditor	0x00000004
Operator	AccessMask must have 0x00000008

If a principal has Enroll, or Officer or Administrator permission, then Read permission is implied (does not need to be explicitly set).

For the CA Operator role defined in [CIMC-PP], a principal must have read permission (implicit or explicit) and must also have the SeBackupPrivilege, specified in [MS-LSAD] section 7, note 37.

For the CA Auditor role defined at [CIMC-PP], a principal must have read permission (implicit or explicit) and must also have the SeSecurityPrivilege, specified in [MS-LSAD] section 7, note 37.

The following table specifies the method name and the list of acceptable permissions required by the caller. Except where mentioned, the caller only needs to possess at least one of these access permissions for the call to be allowed by the CA.

Method name	Acceptable permissions
ICertRequestD::Request	Enroll
ICertRequestD:GetCACert	Enroll
ICertRequestD2::Request2	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertAdminD::GetCRL	Administrator, Officer, Read
ICertAdminD2::GetCAProperty	Administrator, Officer, Read
ICertAdminD2::GetCAPropertyInfo	Administrator, Officer, Read
ICertAdminD::GetViewDefaultColumnSet	Administrator, Officer, Read
ICertAdminD::EnumAttributesOrExtensions	Administrator, Officer, Read
ICertAdminD::OpenView	Administrator, Officer, Read
ICertAdminD::IsValidCertificate	Administrator, Officer, Read
ICertAdminD::GetServerState	None required
ICertAdminD2::GetCASecurity	Administrator, Officer, Read
ICertAdminD2::GetAuditFilter	Administrator, Officer, Read
ICertAdminD2::GetOfficerRights	Administrator, Officer, Read
ICertAdminD2::GetConfigEntry	Administrator, Officer, Read
ICertAdminD2::EnumViewColumnTable	Administrator, Officer, Read
ICertAdminD2::GetMyRoles	Administrator, Officer, Read
ICertAdminD2::GetArchivedKey	Officer
ICertAdminD::SetExtension	Officer
ICertAdminD::SetAttributes	Officer
ICertAdminD::DenyRequest	Officer
ICertAdminD::ReSubmitRequest	Officer

Method name	Acceptable permissions
ICertAdminD::RevokeCertificate	Officer
ICertAdminD::ImportCertificate	Officer
ICertAdmin:D2:ImportKey	Officer
ICertAdminD2::PublishCRLs	Administrator
ICertAdminD::ServerControl	Administrator, Operator
ICertAdminD::Ping	Administrator
ICertAdminD::Ping2	Administrator
ICertAdminD2::SetCASecurity	Administrator
ICertAdminD2::SetCAProperty	Administrator
ICertAdminD2::SetAuditFilter	Administrator, Auditor (either one of these is checked based on a setting on the CA which denotes the permissions to check for SetAuditFilter)
ICertAdminD2::SetOfficerRights	Administrator
ICertAdminD2::SetConfigEntry	Administrator
ICertAdminD2::DeleteRow	Administrator, Officer (both MUST be present)
ICertAdminD::PublishCRL	Administrator
ICertAdminD::BackupPrepare	Operator
ICertAdminD::BackupEnd	Operator
ICertAdminD::RestoreGetDatabaseLocations	Operator
ICertAdminD::BackupGetAttachedInformation	Operator
ICertAdminD::BackupGetBackupLogs	Operator
ICertAdminD::BackupGetDynamicFiles	Operator
ICertAdminD::BackupOpenFile	Operator
ICertAdminD::BackupReadFile	Operator
ICertAdminD::BackupCloseFile	Operator
ICertAdminD::BackupTruncateLogs	Operator

The CA MAY enforce officer rights for any one of the following methods.

- ICertAdminD2::GetArchivedKey
- ICertAdminD::SetExtension
- ICertAdminD::SetAttributes

- ICertAdminD::DenyRequest
- ICertAdminD::ReSubmitRequest
- ICertAdminD::RevokeCertificate

The CA MAY enforce the enrollment agent rights for any one of the following methods.

- ICertRequestD::Request

<20> Section 3.2.1.8: The Microsoft CA keeps this list in a registry multistring value:

```
HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\
{CA Name}\CRLPublicationURLs
```

There is no protocol method specifically to allow manipulation of this list. Instead, the Microsoft CA uses normal registry manipulation tools.

The default values used by the Microsoft CA are a local path on the CA machine

```
{SYSTEM}\CertSrv\CertEnroll\
```

and the AD path

```
ldap: ///CN={CATruncatedName}{CRLNameSuffix},CN={CAServerShortName},
CN=CDP,CN=Public Key Services,CN=Services,DC={contoso},DC=com
```

where:

- {**SYSTEM**} is replaced with the system directory of the CA machine, such as "C:\Windows\System32"
- {**CATruncatedName**} is replaced with the Sanitized Name of the CA, as defined in [MS-WCCE] sections 1.3.2.3 and 3.3.1.
- {**CRLNameSuffix**} is replaced with NULL if the CRL is signed by the first CA key and by "(n)" if the CRL is signed by any subsequent CA key.

  If used, "(n)" replaced with an integer equal to the CA key index minus one.
- **ServerShortName** is replaced with the name of the host on which the CA is running.
- **DC={contoso},DC=com** is replaced with the namespace of the Active Directory domain in which the Microsoft CA is installed.

<21> Section 3.2.5.1: The supported clients are Windows Vista, Windows XP, and Windows 2000 Professional with Admin Pack. The supported servers are Windows Server 2008, Windows Server 2003, and Windows 2000 Server.

<22> Section 3.2.5.1.3: The Windows Server 2003 CA will place 0x80094004 in the *pdwDisposition* parameter and return successfully. The Windows Server 2008 CA will place 0 in the *pdwDisposition* parameter and return 0x80094004 as the error code.

<23> Section 3.2.5.1.6: The Microsoft CA for Windows uses a default clock skew (Config.CA.Clock.Skew.Minutes) of 10 minutes.

<24> Section 3.2.5.1.6: The Microsoft CA computes overlap period based on Config.Base.CRL.Overlap.Period for a base CRL, Config.Delta.CRL.Overlap.Period for a delta CRL as follows:

1. If the registry value for Config.Base.CRL.Overlap.Period is configured and valid (valid means the units are correct and the value is present), this value is used as is. Similarly, if a valid Config.Delta.CRL.Overlap.Period exists, it is used as is.

2. If either one of these values is not present or not valid, the CA uses the following algorithms to determine a value:

   - For a Base CRL:

     1. Interim.Base.CRL.Overlap.Period = MinimumOf(0.1*(Registry.Base.CRL.Validity.Period), 12Hours)

     2. Interim.Base.CRL.Overlap.Period = GreaterOf(Interim.Base.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

     3. Interim.Base.CRL.Overlap.Period = MinimumOf(Interim.Base.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

     4. Config.Base.CRL.Overlap.Period = Interim.Base.CRL.Overlap.Period + Config.CA.Clock.Skew

   - For a Delta CRL:

     1. Interim.Delta.CRL.Overlap.Period = MinimumOf(Registry.Delta.CRL.Validity.Period, 12Hours)

     2. Interim.Delta.CRL.Overlap.Period = GreaterOf(Interim.Delta.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

     3. Interim.Delta.CRL.Overlap.Period = MinimumOf(Interim.Delta.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

     4. Config.Delta.CRL.Overlap.Period = Interim.Delta.CRL.Overlap.Period + Config.CA.Clock.Skew

<25> Section 3.2.5.1.6: The Microsoft CA computes overlap period based on Config.Base.CRL.Overlap.Period for a base CRL, Config.Delta.CRL.Overlap.Period for a delta CRL as follows:

1. If the registry value for Config.Base.CRL.Overlap.Period is configured and valid (valid means the units are correct and the value is present), this value is used as is. Similarly, if a valid Config.Delta.CRL.Overlap.Period exists, it is used as is.

2. If either one of these values is not present or not valid, the CA uses the following algorithms to determine a value:

   - For a Base CRL:

     1. Interim.Base.CRL.Overlap.Period = MinimumOf(0.1*(Registry.Base.CRL.Validity.Period), 12Hours)

2. Interim.Base.CRL.Overlap.Period = GreaterOf(Interim.Base.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

3. Interim.Base.CRL.Overlap.Period = MinimumOf(Interim.Base.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

4. Config.Base.CRL.Overlap.Period = Interim.Base.CRL.Overlap.Period + Config.CA.Clock.Skew

- For a Delta CRL:

1. Interim.Delta.CRL.Overlap.Period = MinimumOf(Registry.Delta.CRL.Validity.Period, 12Hours)

2. Interim.Delta.CRL.Overlap.Period = GreaterOf(Interim.Delta.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

3. Interim.Delta.CRL.Overlap.Period = MinimumOf(Interim.Delta.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

4. Config.Delta.CRL.Overlap.Period = Interim.Delta.CRL.Overlap.Period + Config.CA.Clock.Skew

<26> Section 3.2.5.1.6: The Microsoft CA for Windows uses a default clock skew (Config.CA.Clock.Skew.Minutes) of 10 minutes.

<27> Section 3.2.5.1.6: The Microsoft CA keeps this list in a registry multi-string value:

```
HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\
{CA Name}\CRLPublicationURLs
```

There is no protocol method specifically to allow manipulation of this list. Instead, the Microsoft CA uses normal registry manipulation tools.

The default values used by the Microsoft CA are a local path on the CA machine

```
{SYSTEM}\CertSrv\CertEnroll\
```

and the AD path

```
ldap: ///CN={CAName}{index},CN={CAServerName},CN=CDP,
CN=Public Key Services,CN=Services,DC={contoso},DC=com
```

where:

- **SYSTEM** is replaced with the system directory of the CA machine, such as "C:\Windows\System32"
- **CAName** is replaced with the Sanitized Name of the CA, as defined in [MS-WCCE] sections 1.3.2.3 and 3.3.1.
- **CAServerName** is replaced with the name of the host on which the CA is running.

- **DC={contoso},DC=com** is replaced with the name space of the Active Directory domain in which the Microsoft CA is installed.

<28> Section 3.2.5.1.8: Windows allows serial numbers longer than 20 octets.

<29> Section 3.2.5.1.8: The parameter value 0xfffffffd is only valid on a Windows Server 2008 CA. If this value is used on a Windows Server 2003 CA, the CA fails with return code ERROR_INVALID_PARAMETER (0x80070057).

<30> Section 3.2.5.1.8: The parameter value 0xfffffffe is only valid on a Windows Server 2008 CA. If this value is used on a Windows Server 2003 CA, the CA fails with return code ERROR_INVALID_PARAMETER (0x80070057).

<31> Section 3.2.5.1.8: The Windows mnemonic for reason code 6 is CRL_REASON_CERTIFICATE_HOLD.

<32> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 send the column identifiers as the following **DWORD** array:

```
{0x00001000, 0x00001010, 0x0000100b, 0x00001008, 0x0000100a,
 0x00002003, 0x0000101a, 0x00001022, 0x00001019, 0x00001018,
 0x0000101b,0x0000101c, 0x00001017, 0x00001001}
```

These indexes correspond to the following columns in the Request table:

"Request.RequestId", "Request.RequesterName", "Request.SubmittedWhen", "Request.StatusCode", "Request.DispositionMessage", "CertificateTemplate", "Request.CommonName", "Request.EMail", "Request.OrgUnit", "Request.Organization", "Request.Locality", "Request.State", "Request.Country", "Request.RawRequest"

Windows 2000 Server sends the column identifiers as the following **DWORD** array:

```
{0x00001000, 0x0000100f, 0x00002004, 0x00002006, 0x00002007,
 0x00002011, 0x00002019, 0x00002010, 0x0000200f, 0x00002012
 0x00002013, 0x0000200e, 0x00002001}
```

These indexes correspond to the following columns in the Request table:

"Request.RequestId", "Request.RequesterName", "SerialNumber", "NotBefore", "NotAfter", "CommonName", "EMail", "OrgUnit", "Organization", "Locality", "State", "Country", "RawCertificate"

"Request.RequestId", "Request.RequesterName", "Request.SubmittedWhen", "Request.DispositionMessage", "Request.CommonName", "Request.EMail", "Request.OrgUnit", "Request.Organization", "Request.Locality", "Request.State", "Request.Country", "Request.RawRequest"

<33> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 send the column identifiers as the following **DWORD** array:

```
{0x00001000, 0x00001010, 0x00002006, 0x00002008, 0x00002009,
```

```
 0x00002003, 0x00002015, 0x0000201d, 0x00002014, 0x00002013,
 0x00002016, 0x00002017, 0x00002012, 0x00002001}
```

These indexes correspond to the following columns in the Request table:

"Request.RequestId", "Request.RequesterName", "SerialNumber", "NotBefore", "NotAfter", "CertificateTemplate", "CommonName", "EMail", "OrgUnit","Organization", "Locality", "State", "Country", "RawCertificate"

Windows 2000 Server sends the column identifier as the following **DWORD** array:

```
{0x00001000, 0x0000100f, 0x00002004, 0x00002006, 0x00002007,
 0x00002011, 0x00002019, 0x00002010, 0x0000200f, 0x00002012,
 0x00002013, 0x0000200e, 0x00002001}
```

Request.RequestId", "Request.RequesterName", "SerialNumber", "NotBefore", "NotAfter", "CommonName", "EMail", "OrgUnit","Organization", "Locality", "State", "Country", "RawCertificate"

<34> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 and send the column identifiers as the following **DWORD** array:

```
{0x00001000, 0x00001010, 0x0000100b, 0x00001008, 0x0000100a,
 0x00002003, 0x0000101a, 0x00001022, 0x00001019, 0x00001018,
 0x0000101b, 0x0000101c, 0x00001017, 0x00001001, }
```

These indexes correspond to the following columns in the Request table:

"Request.RequestId", "Request.RequesterName", "Request.SubmittedWhen", "Request.StatusCode", "Request.DispositionMessage", "CertificateTemplate", "Request.CommonName", "Request.EMail", "Request.OrgUnit", "Request.Organization", "Request.Locality", "Request.State", "Request.Country", "Request.RawRequest"

Windows 2000 Server sends the column identifiers the following **DWORD** array:

```
{0x00001000, 0x0000100f, 0x0000100a, 0x00001009, 0x00001017,
 0x0000101f, 0x00001016, 0x00001015, 0x00001018, 0x00001019,
 0x00001014, 0x00001001}
```

These indexes correspond to the following columns in the Request table:

Request.RequestId", "Request.RequesterName", "Request.SubmittedWhen", "Request.DispositionMessage", "Request.CommonName", "Request.EMail", "Request.OrgUnit", "Request.Organization", "Request.Locality", "Request.State", "Request.Country", "Request.RawRequest"

<35> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 and send the column identifiers as the following **DWORD** array:

```
{0x00004000, 0x00004001, 0x00004002, 0x00004003}
```

These indexes correspond to the following columns in the Extension table:

"ExtensionRequestId", "ExtensionName", "ExtensionFlags", "ExtensionRawValue"

Windows 2000 Server returns E_INVALIDARG for this value of the *iColumnSetDefault* parameter.

<36> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 and send the column identifiers as the following **DWORD** array:

```
{0x00003000, 0x00003001, 0x00003002}
```

These indexes correspond to the following columns in the Attribute table:

"AttributeRequestId", "AttributeName", "AttributeValue"

Windows 2000 Server returns E_INVALIDARG for this value of the *iColumnSetDefault* parameter.

<37> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 and send the column identifiers as the following **DWORD** array:

```
{0x00005000, 0x00005001, 0x00005002, 0x00005003, 0x00005004,
 0x00005005, 0x00005006, 0x00005007, 0x00005008, 0x00005009,
 0x0000500a, 0x0000500b, 0x0000500c, 0x0000500d, 0x0000500e,
 0x0000500f, 0x00005010}
```

These indexes correspond to the following columns in the CRL table:

"CRLRowId", "CRLNumber", "CrlMinBase", "CRLNameId", "CrlCount", "CRLThisUpdate", "CRLNextUpdate", "CRLThisPublish", "CRLNextPublish", "CRLEffective", "CRLPropagationComplete", "CRLLastPublish", "CRLPublishAttempts", "CRLPublishFlags", "CRLPublishStatusCode", "CRLPublishError", "CRLRawCRL"

Windows 2000 Server returns E_INVALIDARG for this value of *iColumnSetDefault* parameter.

<38> Section 3.2.5.1.10: Windows Server 2003 and Windows Server 2008 and send the column identifiers as the following **DWORD** array:

```
{0x00001000, 0x00001010, 0x00002006, 0x00002008, 0x00002009,
 0x00002003, 0x00002015, 0x0000201d, 0x00002014,
 0x00002013, 0x00002016, 0x00002017, 0x00002012, 0x00002001,
 0x0000100d, 0x0000100e, 0x0000100f}
```

These indexes correspond to the following columns in the Request table:

"Request.RequestId", "Request.RequesterName", "SerialNumber", "NotBefore", "NotAfter", "CertificateTemplate", "OrgUnit", "DomainComponent", "Organization", "Country", "CommonName",

"Locality", "RawName", "RawCertificate", "Request.RevokedWhen", "Request.RevokedEffectiveWhen", "Request.RevokedReason"

Windows 2000 Server returns E_INVALIDARG for this value of *iColumnSetDefault* parameter.

<39> Section 3.2.5.1.13: Windows Server 2003 returns the total row count. Windows 2000 Server does not.

<40> Section 3.2.5.1.14: If the DCOM connection to a Windows CA on which **OpenView** was called is terminated before a call to **CloseView** is made, Windows Server will eventually release the resources, but may not do so immediately.

<41> Section 3.2.5.1.16: Windows formats return values per the definition of HRESULT as specified in [MS-ERREF]. Negative values indicate errors, positive values indicate success. Specific values are as specified in [MS-ERREF].

<42> Section 3.2.5.1.18: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - ICertAdminD::BackupGetBackupLogs
  - **ICertAdminD::BackupOpenFile**
  - **ICertAdminD::BackupReadFile**
  - **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED ([MS-ERREF], section 2.1) when the following methods are invoked from a remote computer.

- **ICertAdminD::BackupPrepare**

- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<43> Section 3.2.5.1.19: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup-related methods from a remote computer.

The Windows CA enforces this restriction based on the value of the following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - ICertAdminD::BackupGetBackupLogs
  - **ICertAdminD::BackupOpenFile**
  - **ICertAdminD::BackupReadFile**
  - **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer.

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<44> Section 3.2.5.1.20: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - ICertAdminD::BackupGetBackupLogs
  - **ICertAdminD::BackupOpenFile**
  - **ICertAdminD::BackupReadFile**
  - **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer.

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<45> Section 3.2.5.1.21: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - **ICertAdminD::BackupGetBackupLogs**
  - **ICertAdminD::BackupOpenFile**
  - **ICertAdminD::BackupReadFile**

- **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer.

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<46> Section 3.2.5.1.22: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - **ICertAdminD::BackupGetBackupLogs**
  - **ICertAdminD::BackupOpenFile**

- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer.

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<47> Section 3.2.5.1.23: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:
  - **ICertAdminD::BackupGetAttachmentInformation**
  - **ICertAdminD::BackupGetBackupLogs**

- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer:

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<48> Section 3.2.5.1.25: Microsoft CA removes redundant records as recommended in this section.

<49> Section 3.2.5.1.25: In Windows Server 2003 and later versions, the Windows CA defines local configuration to restrict programmatic access to some backup related methods from a remote computer.

The Windows CA enforces this restriction based on the value of following registry key: "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}\InterfaceFlags"

Value	Meaning
0x00000000	The CA will not restrict access to the methods listed for the following servers.
0x00000040	The CA will restrict access to the methods listed for the following servers.

For Windows Server 2003, the CA server can return a failure code when invoking backup methods from a remote computer.

- The CA server can return a failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) for the following methods:
  - **ICertAdminD::BackupPrepare**
  - **ICertAdminD::BackupEnd**
- The CA server can return a failure code of ERROR_UNEXPECTED_ERROR [0x8000FFFF] for the following methods:

- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupTruncateLogs**

For Windows Server 2008, the CA server can return failure code of E_ACCESSDENIED (as specified in [MS-ERREF] section 2.1) when the following methods are invoked from a remote computer:

- **ICertAdminD::BackupPrepare**
- **ICertAdminD::BackupEnd**
- **ICertAdminD::RestoreGetDatabaseLocation**
- **ICertAdminD::BackupGetAttachmentInformation**
- **ICertAdminD::BackupGetBackupLogs**
- **ICertAdminD::BackupGetDynamicFiles**
- **ICertAdminD::BackupOpenFile**
- **ICertAdminD::BackupReadFile**
- **ICertAdminD::BackupCloseFile**
- **ICertAdminD::BackupTruncateLogs**

<50> Section 3.2.5.1.26: The Microsoft CA maintains local configuration to allow or prevent importing of foreign certificates, regardless of the value of *dwFlags*. The configuration is stored in the registry at the location specified below. If the registry value is set to 1, the ImportCertificate method works as documented. If it is set to 0, the FLAG_ALLOW_IMPORT_FOREIGN flag passed as a parameter does not have an effect, and 0x800b0107 is returned.

```
HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}
\KRAFlags (REG_DWORD)
```

<51> Section 3.2.5.1.26: The Microsoft CA maintains local configuration to allow or prevent importing of foreign certificates, regardless of the value of *dwFlags*. The configuration is stored in the registry at the location specified below. If the registry value is set to 1, the ImportCertificate method works as documented. If it is set to 0, the FLAG_ALLOW_IMPORT_FOREIGN flag passed as a parameter does not have an effect, and 0x800b0107 is returned.

```
HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\{CA Name}
\KRAFlags (REG_DWORD)
```

<52> Section 3.2.5.1.26: The Microsoft CA uses DB_DISP_FOREIGN value equal to decimal 12 for "foreign certificate".

<53> Section 3.2.5.2: The supported clients are Windows Vista and Windows XP. The supported servers are Windows Server 2008 and Windows Server 2003.

<54> Section 3.2.5.2.1: The Microsoft CA keeps this list in a registry multi-string value:

```
HKLM\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration\
{CA Name}\CRLPublicationURLs
```

There is no protocol method specifically to allow manipulation of this list. Instead, the Microsoft CA uses normal registry manipulation tools.

The default values used by the Microsoft CA are a local path on the CA machine

```
{SYSTEM}\CertSrv\CertEnroll\
```

and the AD path

```
ldap: ///CN={CAName}{index},CN={CAServerName},CN=CDP,
CN=Public Key Services,CN=Services,DC={contoso},DC=com
```

where:

- **SYSTEM** is replaced with the system directory of the CA machine, such as "C:\Windows\System32"
- **CAName** is replaced with the Sanitized Name of the CA, as defined in [MS-WCCE] sections 1.3.2.3 and 3.3.1.
- **CAServerName** is replaced with the name of the host on which the CA is running.
- **DC={contoso},DC=com** is replaced with the namespace of the Active Directory domain in which the Microsoft CA is installed.

<55> Section 3.2.5.2.1: The Microsoft CA keeps this list in a registry multi-string value, as specified in the preceding Windows Behavior note.

<56> Section 3.2.5.2.1: The Microsoft Windows CA uses a default clock skew (Config.CA.Clock.Skew.Minutes) of 10 minutes.

<57> Section 3.2.5.2.1: The Microsoft CA computes overlap period based on Config.Base.CRL.Overlap.Period for a base CRL, Config.Delta.CRL.Overlap.Period for a delta CRL as follows:

1. If the registry value for Config.Base.CRL.Overlap.Period is configured and valid (valid means the units are correct and the value is present), this value is used as is. Similarly, if a valid Config.Delta.CRL.Overlap.Period exists, it is used as is.
2. If either one of these values is not present or not valid, the CA uses the following algorithms to determine a value:
   - For a Base CRL:

1. Interim.Base.CRL.Overlap.Period = MinimumOf(0.1*(Registry.Base.CRL.Validity.Period), 12Hours)

2. Interim.Base.CRL.Overlap.Period = GreaterOf(Interim.Base.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

3. Interim.Base.CRL.Overlap.Period = MinimumOf(Interim.Base.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

4. Config.Base.CRL.Overlap.Period = Interim.Base.CRL.Overlap.Period + Config.CA.Clock.Skew

- For a Delta CRL:

1. Interim.Delta.CRL.Overlap.Period = MinimumOf(Registry.Delta.CRL.Validity.Period, 12Hours)

2. Interim.Delta.CRL.Overlap.Period = GreaterOf(Interim.Delta.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

3. Interim.Delta.CRL.Overlap.Period = MinimumOf(Interim.Delta.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

4. Config.Delta.CRL.Overlap.Period = Interim.Delta.CRL.Overlap.Period + Config.CA.Clock.Skew

<58> Section 3.2.5.2.1: The Microsoft CA computes overlap period based on Config.Base.CRL.Overlap.Period for a base CRL, Config.CRL.Delta.Overlap.Period for a delta CRL

<59> Section 3.2.5.2.1: The Microsoft CA computes overlap period based on Config.Base.CRL.Overlap.Period for a base CRL, Config.Delta.CRL.Overlap.Period for a delta CRL as follows:

1. If the registry value for Config.Base.CRL.Overlap.Period is configured and valid (valid means the units are correct and the value is present), this value is used as is. Similarly, if a valid Config.Delta.CRL.Overlap.Period exists, it is used as is.

2. If either one of these values is not present or not valid, the CA uses the following algorithms to determine a value:

   - For a Base CRL:

   1. Interim.Base.CRL.Overlap.Period = MinimumOf(0.1*(Registry.Base.CRL.Validity.Period), 12Hours)

   2. Interim.Base.CRL.Overlap.Period = GreaterOf(Interim.Base.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

   3. Interim.Base.CRL.Overlap.Period = MinimumOf(Interim.Base.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

   4. Config.Base.CRL.Overlap.Period = Interim.Base.CRL.Overlap.Period + Config.CA.Clock.Skew

   - For a Delta CRL:

   1. Interim.Delta.CRL.Overlap.Period = MinimumOf(Registry.Delta.CRL.Validity.Period, 12Hours)

2. Interim.Delta.CRL.Overlap.Period = GreaterOf(Interim.Delta.CRL.Overlap.Period, 1.5*(Config.CA.Clock.Skew))

3. Interim.Delta.CRL.Overlap.Period = MinimumOf(Interim.Delta.CRL.Overlap.Period, Registry.Base.CRL.Validity.Period)

4. Config.Delta.CRL.Overlap.Period = Interim.Delta.CRL.Overlap.Period + Config.CA.Clock.Skew

<60> Section 3.2.5.2.1: The Microsoft CA keeps this list in a registry multi-string value, as specified in this section in a preceding Windows Behavior note.

<61> Section 3.2.5.2.5: This rule applies only to a Windows Server 2008 CA. A Windows 2000 or Windows Server 2003 CA will not enforce that *cColumn* is greater than 0. Rather, when *cColumn* is equal to zero, it will set *pcColumn* equal to zero, *pctbColumnInfo->cb* equal to 0, *pctbColumnInfo->pb* will point to a zero-length item, and the function will return successfully.

<62> Section 3.2.5.2.7: A Windows CA defines six permissions: Enroll, Read, Officer, Administrator, Operator, and Auditor.

For CA security (GetCASecurity, SetCASecurity, and GetMyRoles), the Microsoft Certificate Authority assigns permissions to principals (identified by the Access Control Entry (ACE)) in the following manner.

Permission	Bit value
Read	0x00000100
Enroll	0x00000200
Officer	0x00000002
Administrator	0x00000001
Auditor	0x00000004
Operator	AccessMask must have 0x00000008

If a principal has Enroll, Officer, or Administrator permission, then Read permission is implied (does not need to be explicitly set).

For the CA Operator role defined in [CIMC-PP], a principal must have read permission (implicit or explicit) and must also have the SeBackupPrivilege, as specified in [MS-LSAD] section 7, note 37.

For the CA Auditor role defined at [CIMC-PP], a principal must have Read permission (implicit or explicit) and must also have the SeSecurityPrivilege, as specified in [MS-LSAD] section 7, note 37.

The following table specifies the method name and the list of acceptable permissions required by the caller. Except where mentioned, the caller needs only to possess at least one of these access permissions for the call to be allowed by the CA.

Method name	Acceptable permissions
ICertRequestD::Request	Enroll
ICertRequestD:GetCACert	Enroll

Method name	Acceptable permissions
ICertRequestD2::Request2	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertAdminD::GetCRL	Administrator, Officer, Read
ICertAdminD2::GetCAProperty	Administrator, Officer, Read
ICertAdminD2::GetCAPropertyInfo	Administrator, Officer, Read
ICertAdminD::GetViewDefaultColumnSet	Administrator, Officer, Read
ICertAdminD::EnumAttributesOrExtensions	Administrator, Officer, Read
ICertAdminD::OpenView	Administrator, Officer, Read
ICertAdminD::IsValidCertificate	Administrator, Officer, Read
ICertAdminD::GetServerState	None required
ICertAdminD2::GetCASecurity	Administrator, Officer, Read
ICertAdminD2::GetAuditFilter	Administrator, Officer, Read
ICertAdminD2::GetOfficerRights	Administrator, Officer, Read
ICertAdminD2::GetConfigEntry	Administrator, Officer, Read
ICertAdminD2::EnumViewColumnTable	Administrator, Officer, Read
ICertAdminD2::GetMyRoles	Administrator, Officer, Read
ICertAdminD2::GetArchivedKey	Officer
ICertAdminD::SetExtension	Officer
ICertAdminD::SetAttributes	Officer
ICertAdminD::DenyRequest	Officer
ICertAdminD::ReSubmitRequest	Officer
ICertAdminD::RevokeCertificate	Officer
ICertAdminD::ImportCertificate	Officer
ICertAdmin:D2:ImportKey	Officer
ICertAdminD2::PublishCRLs	Administrator
ICertAdminD::ServerControl	Administrator, Operator

Method name	Acceptable permissions
ICertAdminD::Ping	Administrator
ICertAdminD::Ping2	Administrator
ICertAdminD2::SetCASecurity	Administrator
ICertAdminD2::SetCAProperty	Administrator
ICertAdminD2::SetAuditFilter	Administrator, Auditor (either one of these is checked based on a setting on the CA which denotes the permissions to check for SetAuditFilter)
ICertAdminD2::SetOfficerRights	Administrator
ICertAdminD2::SetConfigEntry	Administrator
ICertAdminD2::DeleteRow	Administrator, Officer (both MUST be present)
ICertAdminD::PublishCRL	Administrator
ICertAdminD::BackupPrepare	Operator
ICertAdminD::BackupEnd	Operator
ICertAdminD::RestoreGetDatabaseLocations	Operator
ICertAdminD::BackupGetAttachedInformation	Operator
ICertAdminD::BackupGetBackupLogs	Operator
ICertAdminD::BackupGetDynamicFiles	Operator
ICertAdminD::BackupOpenFile	Operator
ICertAdminD::BackupReadFile	Operator
ICertAdminD::BackupCloseFile	Operator
ICertAdminD::BackupTruncateLogs	Operator

The CA MAY enforce officer rights for any one of the following methods.

- ICertAdminD2::GetArchivedKey
- ICertAdminD::SetExtension
- ICertAdminD::SetAttributes
- ICertAdminD::DenyRequest
- ICertAdminD::ReSubmitRequest
- ICertAdminD::RevokeCertificate

The CA MAY enforce the enrollment agent rights for the following method.

- ICertRequestD::Request

<63> Section 3.2.5.2.17: A Windows CA defines six permissions: Enroll, Read, Officer, Administrator, Operator, and Auditor.

For CA security (GetCASecurity, SetCASecurity, and GetMyRoles), the Microsoft Certificate Authority assigns permissions to principals (identified by the Access Control Entry (ACE)) in the following manner.

Permission	Bit value
Read	0x00000100
Enroll	0x00000200
Officer	0x00000002
Administrator	0x00000001
Auditor	0x00000004
Operator	AccessMask must have 0x00000008

If a principal has Enroll, Officer, or Administrator permission, then Read permission is implied (does not need to be explicitly set).

For the CA Operator role defined in [CIMC-PP], a principal must have read permission (implicit or explicit) and must also have the SeBackupPrivilege, as specified in [MS-LSAD] section 7, note 37.

For the CA Auditor role defined in [CIMC-PP], a principal must have Read permission (implicit or explicit) and must also have the SeSecurityPrivilege, as specified in [MS-LSAD] section 7, note 37.

The following table specifies the method name and the list of acceptable permissions required by the caller. Except where mentioned, the caller needs only to possess at least one of these access permissions for the call to be allowed by the CA.

Method name	Acceptable permissions
ICertRequestD::Request	Enroll
ICertRequestD:GetCACert	Enroll
ICertRequestD2::Request2	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertRequestD2::GetCAProperty	Enroll
ICertRequestD2::GetCAPropertyInfo	Enroll
ICertAdminD::GetCRL	Administrator, Officer, Read
ICertAdminD2::GetCAProperty	Administrator, Officer, Read
ICertAdminD2::GetCAPropertyInfo	Administrator, Officer, Read
ICertAdminD::GetViewDefaultColumnSet	Administrator, Officer, Read

Method name	Acceptable permissions
ICertAdminD::EnumAttributesOrExtensions	Administrator, Officer, Read
ICertAdminD::OpenView	Administrator, Officer, Read
ICertAdminD::IsValidCertificate	Administrator, Officer, Read
ICertAdminD::GetServerState	None required
ICertAdminD2::GetCASecurity	Administrator, Officer, Read
ICertAdminD2::GetAuditFilter	Administrator, Officer, Read
ICertAdminD2::GetOfficerRights	Administrator, Officer, Read
ICertAdminD2::GetConfigEntry	Administrator, Officer, Read
ICertAdminD2::EnumViewColumnTable	Administrator, Officer, Read
ICertAdminD2::GetMyRoles	Administrator, Officer, Read
ICertAdminD2::GetArchivedKey	Officer
ICertAdminD::SetExtension	Officer
ICertAdminD::SetAttributes	Officer
ICertAdminD::DenyRequest	Officer
ICertAdminD::ReSubmitRequest	Officer
ICertAdminD::RevokeCertificate	Officer
ICertAdminD::ImportCertificate	Officer
ICertAdmin:D2:ImportKey	Officer
ICertAdminD2::PublishCRLs	Administrator
ICertAdminD::ServerControl	Administrator, Operator
ICertAdminD::Ping	Administrator
ICertAdminD::Ping2	Administrator
ICertAdminD2::SetCASecurity	Administrator
ICertAdminD2::SetCAProperty	Administrator
ICertAdminD2::SetAuditFilter	Administrator, Auditor (either one of these is checked, based on a setting on the CA which denotes the permissions to check for SetAuditFilter)
ICertAdminD2::SetOfficerRights	Administrator
ICertAdminD2::SetConfigEntry	Administrator
ICertAdminD2::DeleteRow	Administrator, Officer (both MUST be present)

Method name	Acceptable permissions
ICertAdminD::PublishCRL	Administrator
ICertAdminD::BackupPrepare	Operator
ICertAdminD::BackupEnd	Operator
ICertAdminD::RestoreGetDatabaseLocations	Operator
ICertAdminD::BackupGetAttachedInformation	Operator
ICertAdminD::BackupGetBackupLogs	Operator
ICertAdminD::BackupGetDynamicFiles	Operator
ICertAdminD::BackupOpenFile	Operator
ICertAdminD::BackupReadFile	Operator
ICertAdminD::BackupCloseFile	Operator
ICertAdminD::BackupTruncateLogs	Operator

The CA MAY enforce Officer rights for any one of the following methods.

- ICertAdminD2::GetArchivedKey
- ICertAdminD::SetExtension
- ICertAdminD::SetAttributes
- ICertAdminD::DenyRequest
- ICertAdminD::ReSubmitRequest
- ICertAdminD::RevokeCertificate

The CA MAY enforce the enrollment agent rights for any one of the following methods.

- ICertRequestD::Request

<64> Section 3.2.5.2.18: If the CA fails to delete all rows that match a date restriction as specified above, the Windows CA returns an HRESULT value of ERROR_OUT_OF_MEMORY to indicate to the client that more rows matching the criteria may remain.

8 Index

K

L

M

N

O

P

R

S

T

V

W