IEC 62541-12

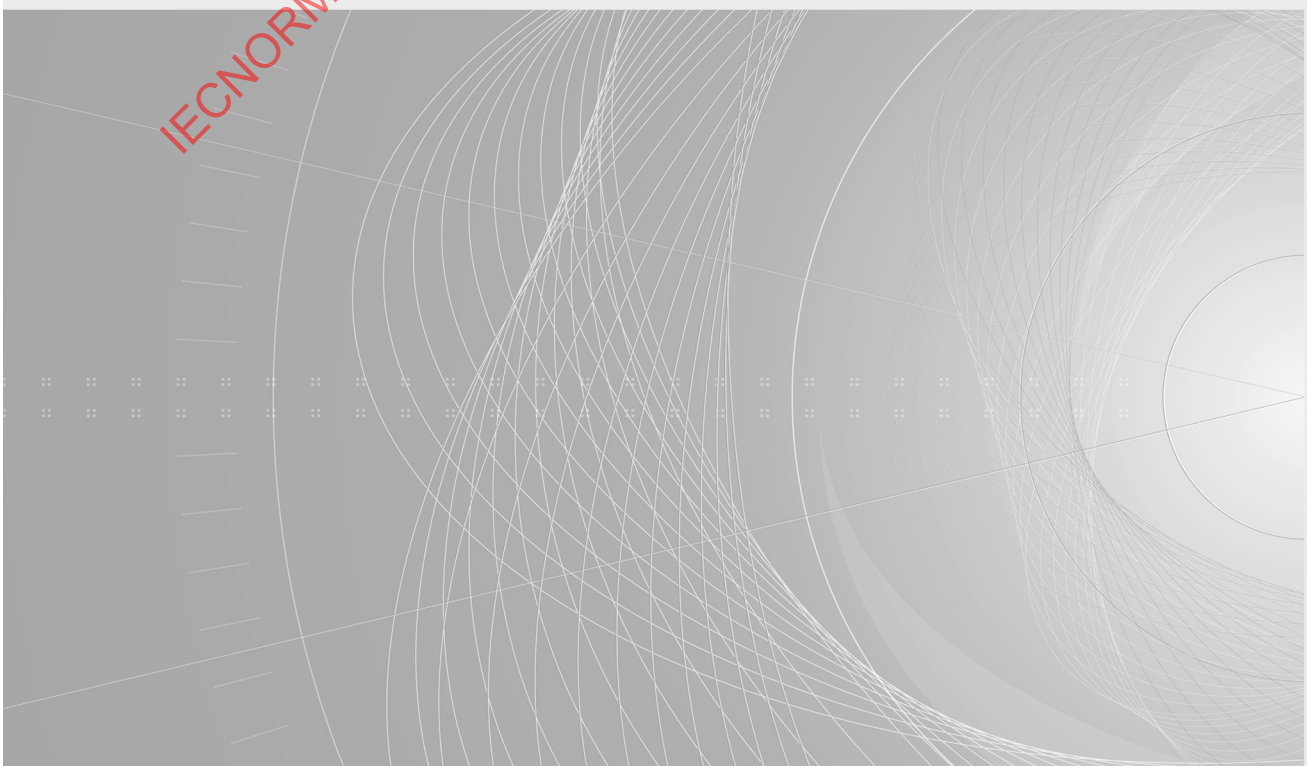
Edition 1.0 2020-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

OPC unified architecture – Part 12: Discovery and global services

Architecture unifiée OPC – Partie 12: Services globaux et de découverte



IEC 62541-12:2020-06(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Recherche de publications IEC - webstore.iec.ch/advsearchform
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et une fois par mois par email.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: sales@iec.ch.

Electropedia - www.electropedia.org
Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 000 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 16 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
67 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

IEC 62541-12

Edition 1.0 2020-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

OPC unified architecture – Part 12: Discovery and global services

Architecture unifiée OPC – Partie 12: Services globaux et de découverte

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40

ISBN 978-2-8322-8455-1

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

OPC UNIFIED ARCHITECTURE –

Part 12: Discovery and global services

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International standard IEC 62541-12 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this standard is based on the following documents:

FDIS	Report on voting
65E/711/FDIS	65E/723/RVD

Full information on the voting for the approval of this International Standard can be found in the report on voting indicated in the above table.

This document has been drafted in accordance with the ISO/IEC Directives, Part 2.

Throughout this document and the other parts of the IEC 62541 series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in the "Terms and definition" clause in one of the parts of the IEC 62541 series.

Italics are also used to denote the name of a service input or output parameter or the name of a structure or element of a structure that are usually defined in tables.

The *italicized terms and names* are, with a few exceptions, written in camel-case (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). For example, the defined term is AddressSpace instead of Address Space. This makes it easier to understand that there is a single definition for AddressSpace, not separate definitions for Address and Space.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under "http://webstore.iec.ch" in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

OPC UNIFIED ARCHITECTURE –

Part 12: Discovery and global services

1 Scope

This part of IEC 62541 specifies how OPC Unified Architecture (OPC UA) *Clients* and *Servers* interact with *DiscoveryServers* when used in different scenarios. It specifies the requirements for the *LocalDiscoveryServer, LocalDiscoveryServer-ME and GlobalDiscoveryServer*. It also defines information models for *Certificate* management, *KeyCredential* management and *Authorization Services*.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and concepts*

IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model*

IEC 62541-3, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4, *OPC Unified Architecture – Part 4: Services*

IEC 62541-5, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

IEC 62541-9, *OPC Unified Architecture – Part 9: Alarms and conditions*

IEC 62541-14, *OPC Unified Architecture – Part 14: PubSub*

X.500: ISO/IEC 9594-1:2017, *Information technology – Open Systems Interconnection – The Directory – Part 1: Overview of concepts, models and services*

IETF RFC 1035, *DNS-Name: Domain Names – Implementation and Specification*
http://www.ietf.org/rfc/rfc1035.txt

IETF RFC 2986, *PKCS #10: Certification Request Syntax Specification*
http://www.ietf.org/rfc/rfc2986.txt

IETF RFC 3927, *Auto-IP: Dynamic Configuration of IPv4 Link-Local Addresses*
http://www.ietf.org/rfc/rfc3927.txt

IETF RFC 5958, *Asymmetric Key Packages*
http://www.ietf.org/rfc/rfc5958.txt

IETF RFC 6762, *mDNS: Multicast DNS*
http://www.ietf.org/rfc/rfc6762.txt

IETF RFC 6763, *DNS-SD: DNS Based Service Discovery*
http://www.ietf.org/rfc/rfc6763.txt

IETF RFC 7030: *Enrollment over Secure Transport*
http://www.ietf.org/rfc/rfc7030.txt

PKCS #12: *Personal Information Exchange Syntax*
http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11301-wp-pkcs-12v1-1-personal-information-exchange-syntax.pdf

DI: *OPC Unified Architecture for Devices (DI)*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-devices-di/

ADI: *OPC Unified Architecture for Analyzer Devices (ADI)*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-analyzer-devices-adi/

PLCopen: *OPC Unified Architecture / PLCopen Information Model*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-plcopen-information-model/

FDI: *OPC Unified Architecture for FDI*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-fdi/

ISA-95: *ISA-95 Common Object Model*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/isa-95-common-object-model/

3 Terms, definitions, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC TR 62541-2, IEC 62541-3, IEC 62541-4, IEC 62541-6 and IEC 62541-9 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1.1
Certificate Group
context used to describe the *Trust List* and *Certificate(s)* associated with an *Application*

3.1.2
Certificate Request
PKCS #10 encoded structure used to request a new *Certificate* from a *Certificate Authority*

3.1.3
KeyCredential
unique identifier and a secret used to access a *Server*, an *Authorization Service* or a *Broker*

Note 1 to entry: A user name and password is an example of a credential.

3.1.4
KeyCredentialService
software application that provides *KeyCredentials* needed to access a *Server*, an *Authorization Service* or a *Broker*

3.1.5
DirectoryService
software application, or a set of applications, that stores and organizes information about resources such as computers or services

3.1.6
DiscoveryServer
Application that maintains a list of OPC UA *Servers* that are available on the network and provides mechanisms for Clients to obtain this list

3.1.7
DiscoveryUrl
URL for a network *Endpoint* that provides the information required to connect to a *Client* or *Server*

3.1.8
GlobalDiscoveryServer
GDS
DiscoveryServer that maintains a list of OPC UA *Applications* available in an administrative domain

Note 1 to entry: A GDS can also provide certificate management services.

3.1.9
IPAddress
unique number assigned to a network interface that allows Internet Protocol (IP) requests to be routed to that interface

Note 1 to entry: An *IPAddress* for a host can change over time.

3.1.10
LocalDiscoveryServer
LDS
DiscoveryServer that maintains a list of all *Servers* that have registered with it

Note 1 to entry: *Servers* normally register with the LDS on the same host.

3.1.11
LocalDiscoveryServer-ME
LDS-ME
LocalDiscoveryServer that includes the MulticastExtension

3.1.12
MulticastExtension
extension to a *LocalDiscoveryServer* that adds support for the *mDNS* protocol

3.1.13
MulticastSubnet
network that allows multicast packets to be sent to all nodes connected to the network

Note 1 to entry: A *MulticastSubnet* is not necessarily the same as a TCP/IP subnet.

3.1.14
Network Service
secured resource on a network that provides functionality used by *Clients* and/or *Servers*

Note 1 to entry: An Authorization Service (AS) is an example of a Network Service.

3.1.15
ServerCapabilityIdentifier
short identifier which uniquely identifies a set of discoverable capabilities supported by a *Server*

Note 1 to entry: The list of the currently defined *ServerCapabilityIdentifiers* is in Annex D.

3.2 Abbreviated terms and symbols

API	application programming interface
CA	certificate authority
CRL	certificate revocation list
CSR	certificate signing request
DER	distinguished encoding rules
DNS	Domain Name System
EST	Enrolment over Secure Transport
GDS	Global Discovery Server
IANA	Internet Assigned Numbers Authority
LDAP	Lightweight Directory Access Protocol
LDS	Local Discovery Server
LDS-ME	Local Discovery Server with Multicast Extension
mDNS	Multicast Domain Name System
NAT	network address translation
PEM	Privacy-Enhanced Mail
PFX	Personal Information Exchange
PKCS	Public-Key Cryptography Standards
SHA1	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UA	Unified Architecture
UDDI	Universal Description, Discovery And Integration

3.3 Conventions for namespaces

This document uses multiple namespaces to define *Nodes*. The following abbreviated terms are used in the definitions for these *Nodes*:

CORE	http://opcfoundation.org/UA/
GDS	http://opcfoundation.org/UA/GDS/

The default namespace for each *Node* is defined at the top of the table. All of the *BrowseNames* in the table use the default namespace unless the *BrowseName* is preceded by one of the above abbreviations.

The *NamespaceMetadataType Object* for the GDS namespace is defined in Table 1.

Table 1 – GDS NamespaceMetadataType Object definition

Attribute	Value		
BrowseName	http://opcfoundation.org/UA/GDS/		
Namespace	GDS (see 3.3)		
TypeDefinition	NamespaceMetadataType defined in IEC 62541-5.		
References	**NodeClass**	**BrowseName**	**Value**
HasProperty	Variable	NamespaceUri	http://opcfoundation.org/UA/GDS/
HasProperty	Variable	NamespaceVersion	1.04
HasProperty	Variable	NamespacePublicationDate	2016-12-31

4 The discovery process

4.1 Overview

The discovery process allows applications to find other applications on the network and then discover how to connect to them. Note that this discussion builds on the discovery related concepts defined in IEC 62541-4. Discoverable applications are generally *Servers*, however, some *Clients* will support reverse connections as described in IEC 62541-6 and want *Servers* to be able to discover them.

Clients and *Servers* can be on the same host, on different hosts in the same subnet, or even on completely different locations in an administrative domain. The following clauses describe the different configurations and how discovery can be accomplished.

The mechanisms for *Clients* to discover *Servers* are specified in 4.3.

The mechanisms for *Servers* to make themselves discoverable are specified in 4.2.

The *Discovery Services* are specified in IEC 62541-4. They are implemented by individual *Servers* and by dedicated *DiscoveryServers*. The following dedicated *DiscoveryServers* provide a way for applications to discover registered OPC UA applications in different situations:

- a *LocalDiscoveryServer* (LDS) maintains discovery information for all applications that have registered with it, usually all applications available on the host that it runs on;
- a *LocalDiscoveryServer* with the *MulticastExtension* (LDS-ME) maintains discovery information for all applications that have been announced on the local *MulticastSubnet*;
- a *GlobalDiscoveryServer* (GDS) maintains discovery information for applications available in an administrative domain.

LDS and LDS-ME are specified in Clause 5. The GDS is specified in Clause 6.

Annex A provides guidelines on the use of firewalls in discovery. Annex B defines the Global Discovery NodeSet. Annex F defines a directory layout for applications to store their *Certificates* on a file system. Annex G provides instructions for the application installation procedure. Annex H describes how Certificate management described in this document compares to RFC7030.

4.2 Registration and announcement of Applications

4.2.1 Overview

This subclause describes how an application registers itself so it can be discovered. Most *Applications* will want other applications to discover them. *Applications* that do not wish to be discovered openly should not register with a *DiscoveryServer*. In this case, such *Applications* should only publish a *DiscoveryUrl* via some out-of-band mechanism to be discovered by specific *Applications*.

4.2.2 Hosts with a LocalDiscoveryServer

Applications register themselves with the LDS on the same host if they wish to be discovered. The registration ensures that the applications is visible for local discovery (see 4.3.4) and *MulticastSubnet* discovery if the LDS is an LDS-ME (see 4.3.5).

The OPC UA Standard (IEC 62541-4) defines a *RegisterServer2 Service*, which provides additional registration information. All *Applications* and the *LocalDiscoveryServer* shall support the *RegisterServer2 Service* and, for backwards compatibility, the older *RegisterServer Service*. If an *Application* encounters an older LDS that returns a *Bad_ServiceUnsupported* error when calling *RegisterServer2 Service*, it shall try again with *RegisterServer Service*.

The *RegisterServer2 Service* allows the *Application* to specify zero or more *ServerCapability Identifiers*. *ServerCapabilityIdentifiers* are short, string identifiers of well-known OPC UA features. *Applications* can use these identifiers as a filter during discovery.

The set of known *ServerCapabilityIdentifiers* is specified in Annex D and is limited to features that are considered to be important enough to report before an application makes a connection. For example, support for the GDS information model or the Alarms information model are *Server* capabilities that have a *ServerCapabilityIdentifier* defined.

Before an application registers with the LDS, it should call the *GetEndpoints Service* and choose the most secure endpoint supported by the LDS and then call *RegisterServer2* or *RegisterServer*.

Registration with LDS or LDS-ME is illustrated in Figure 1.

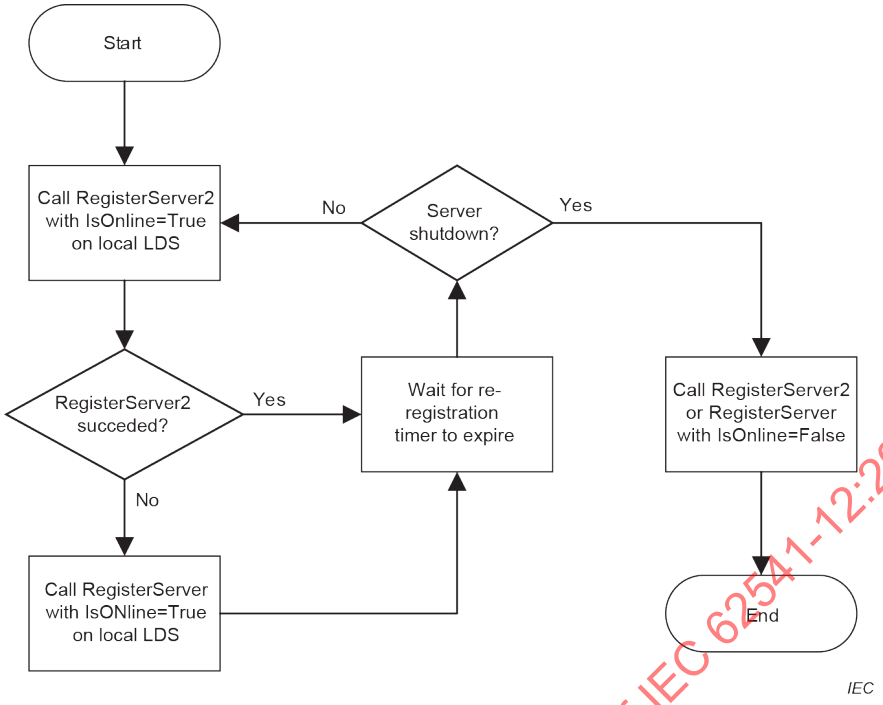


Figure 1 – The Registration process with an LDS

See IEC 62541-4 for more information on the re-registration timer and the *IsOnline* flag.

4.2.3 Hosts without a LocalDiscoveryServer

Dedicated systems (usually embedded systems) with exactly one *Server* installed may not have a separate LDS. Such *Servers* shall become their own LDS or LDS-ME by implementing *FindServers* and *GetEndpoints Services* at the well-known address for an LDS. They should also announce themselves on the *MulticastSubnet* with a basic *MulticastExtension*. This requires a small subset of an mDNS Responder (see *mDNS* and Annex C) that announces the *Server* and responds to mDNS probes. The *Server* may not provide the caching and address resolution implemented by a full mDNS Responder.

4.3 The discovery process for Clients to find Servers

4.3.1 Overview

The discovery process allows *Clients* to find *Servers* on the network and then discover how to connect to them. Once a *Client* has this information, it can save it and use it to connect directly to the *Server* again without going through the discovery process. *Clients* that cannot connect with the saved connection information should assume the *Server* configuration has changed and therefore repeat the discovery process.

A *Client* has several choices for finding *Servers*:

- out-of-band discovery (i.e. entry into a GUI) of a *DiscoveryUrl* for a *Server*;
- calling *FindServers* on the LDS installed on the *Client* host;
- calling *FindServers* on a remote LDS, where the *HostName* for the remote host is manually entered;
- calling *FindServersOnNetwork* (see IEC 62541-4) on the LDS-ME installed on *Client* host;
- supporting the LDS-ME functionality locally in the Client;
- searching for *Servers* known to a *GlobalDiscoveryServer*.

The *DiscoveryUrl* provides all of the information a *Client* needs to connect to a *DiscoveryEndpoint* (see 4.3.3).

4.3.2 Security

Clients should be aware of rogue *DiscoveryServers* that might direct them to rogue *Servers*. *Clients* can use the SSL/TLS server certificate (if available) to verify that the *DiscoveryServer* is a server that they trust and/or ensure that they trust any *Server* provided by the *DiscoveryServer*. See IEC TR 62541-2 for a detailed discussion of these issues.

4.3.3 Simple Discovery with a DiscoveryUrl

Every *Server* has one or more *DiscoveryUrls* that allow access to its *Endpoints*. Once a *Client* obtains (e.g. via manual entry into a form) the *DiscoveryUrl* for the *Server,* it reads the *EndpointDescriptions* using the *GetEndpoints Service* defined in IEC 62541-4.

The discovery process for this scenario is illustrated in Figure 2.

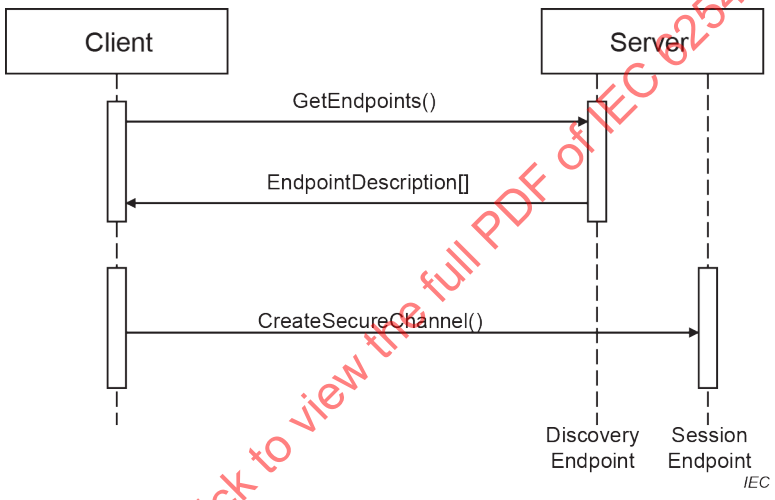


Figure 2 – The simple Discovery process

4.3.4 Local Discovery

In many cases, *Clients* do not know which *Servers* exist but possibly know which hosts can have *Servers* on them. In this situation, the *Client* will look for the *LocalDiscoveryServer* on a host by constructing a *DiscoveryUrl* using the well-known addresses defined in IEC 62541-6.

If a *Client* finds a *LocalDiscoveryServer* then it will call the *FindServers Service* on the LDS to obtain a list of *Servers* and their *DiscoveryUrls*. The *Client* would then call the *GetEndpoints* service for one of the *Servers* returned. The discovery process for this scenario is illustrated in Figure 3.

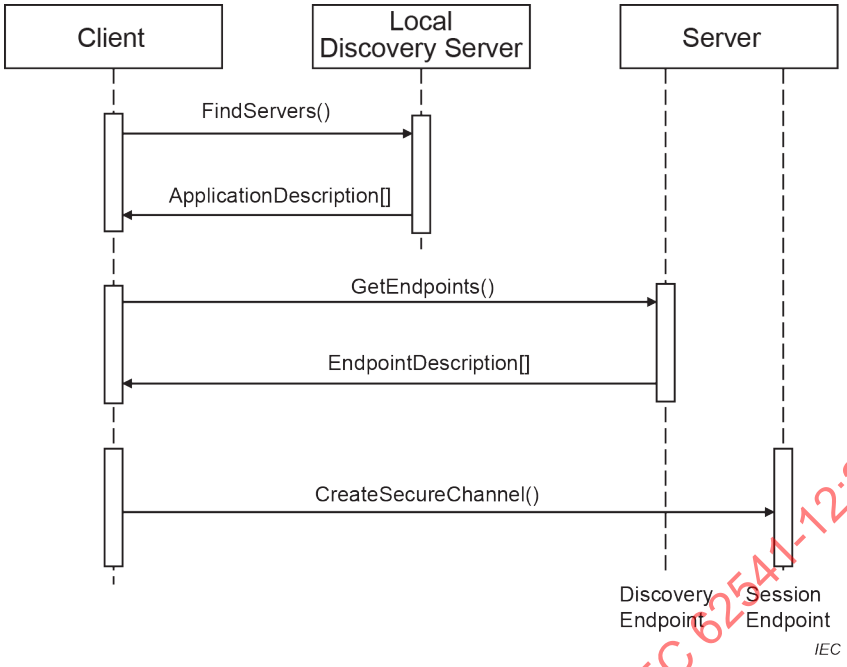


Figure 3 – The Local Discovery process

4.3.5 MulticastSubnet Discovery

In some situations *Clients* will not know which hosts have *Servers*. In these situations the *Client* will look for a *LocalDiscoveryServer* with the *MulticastExtension* on its local host and requests a list of *DiscoveryUrls* for *Servers* and *DiscoveryServers* available on the *MulticastSubnet*.

The discovery process for this scenario is illustrated in Figure 4.

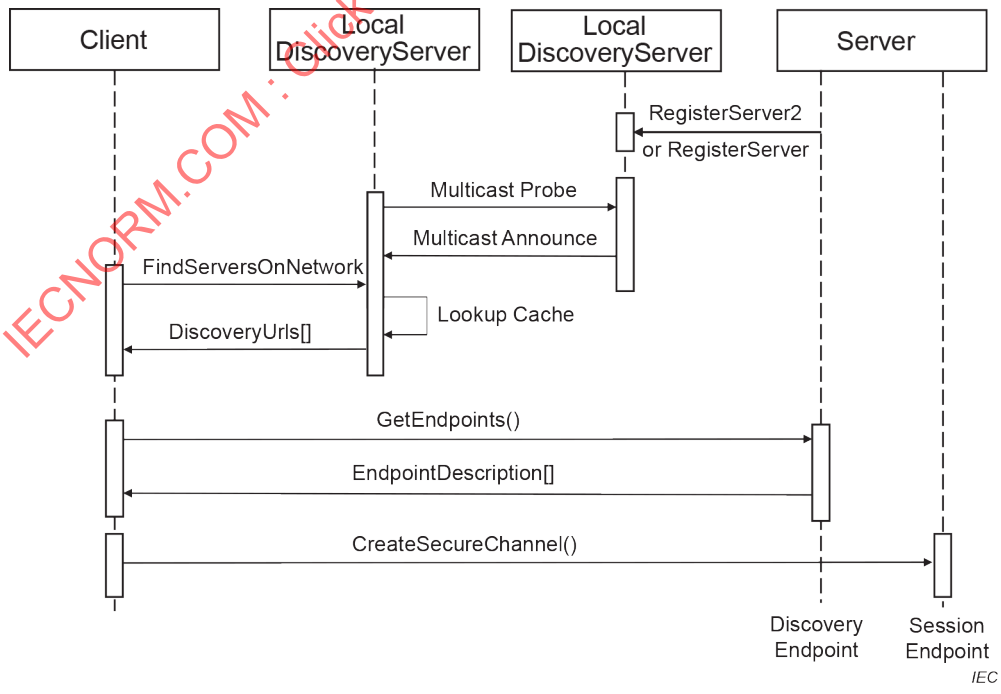


Figure 4 – The MulticastSubnet Discovery process

In this scenario, the *Server* uses the *RegisterServer2 Service* to tell a *LocalDiscoveryServer* to announce the *Server* on the *MulticastSubnet*. The *Client* will receive the *DiscoveryUrl* and *ServerCapabilityIdentifiers* for the Server when it calls *FindServersOnNetwork* and then connects directly to the *Server*. When a *Client* calls *FindServers* it only receives the *Servers* running on the same host as the LDS.

Clients running on embedded systems may not have a LDS-ME available on the system. These *Clients* can support an mDNS Responder that understands how OPC UA concepts are mapped to mDNS messages and maintains the same table of *Servers* as maintained by the LDS-ME. This mapping is described in Annex C.

4.3.6 Global Discovery

A GDS is an OPC UA *Server* that allows *Clients* to search for *Servers* in the administrative domain. It may also provide Certificate Services (see Clause 7). It provides *Methods* that allow applications to search for other applications (See Clause 6). To access the GDS, the *Client* will create a *Session* with the *GDS* and use the *Call* service to invoke the *QueryApplications Method* (see 6.3.11). The *QueryServers Method* is similar to the *FindServers* service except that it provides more advanced search and filter criteria. The discovery process is illustrated in Figure 5.

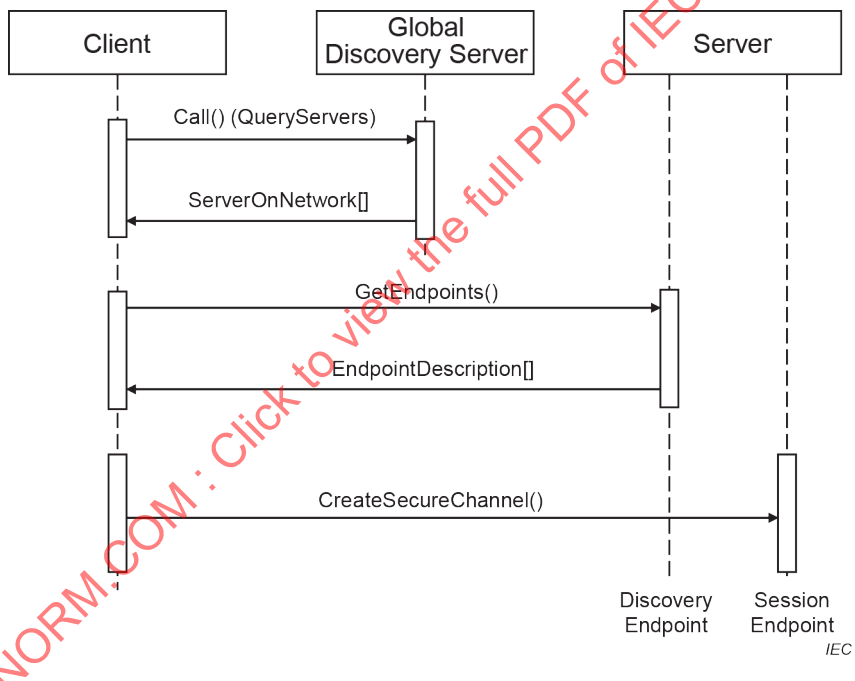


Figure 5 – The Global Discovery process

The GDS may be coupled with any of the previous network architectures. For each *MulticastSubnet,* one or more LDSs may be registered with a GDS.

The *Client* can also be configured with the URL of the GDS using an out of band mechanism.

The complete discovery process is shown in Figure 6.

4.3.7 Combined Discovery Process for Clients

The use cases in the preceding clauses imply a number of choices that have to be made by *Clients* when a *Client* needs to connect to a *Server*. These choices are combined together in Figure 6.

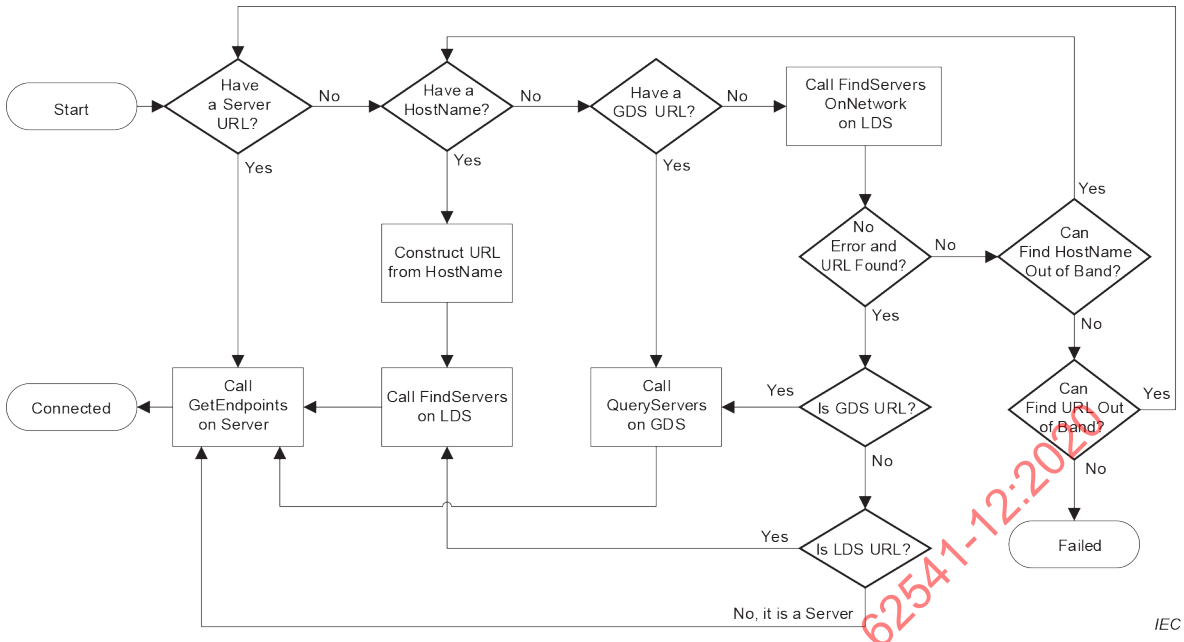


Figure 6 – The Discovery Process for Clients

FindServersOnNetwork can be called on the local LDS, however, it can also be called on a remote LDS which is part of a different *MulticastSubnet*.

An out-of-band mechanism is a way to find a URL or a *HostName* that is not described by this document. For example, a user could manually enter a URL or use system specific APIs to browse the network neighbourhood.

A *Client* that goes through the discovery process can save the URL that was discovered. If the *Client* restarts later, it can use that URL and bypass the discovery process. If reconnection fails, the *Client* will have to go through the process again.

5 Local Discovery Server

5.1 Overview

Each host that could have multiple discoverable applications installed should have a standalone *LocalDiscoveryServer* installed. The *LocalDiscoveryServer* shall expose one or more *Endpoints* which support the *FindServers* and *GetEndpoints* services defined in IEC 62541-4 for all applications on the host. In addition, the *LocalDiscoveryServer* shall provide at least one *Endpoint* which implements the *RegisterServer* service for these applications.

In systems (usually embedded systems) with exactly one *Server* installed this *Server* may also be the LDS (see 4.2.3).

An LDS-ME will announce all applications that it knows about on the local *MulticastSubnet*. In order to support this, a *LocalDiscoveryServer* supports the *RegisterServer2 Service* defined in IEC 62541-4. For backward compatibility, a *LocalDiscoveryServer* also supports the *RegisterServer Service* which is defined in IEC 62541-4.

Each host with OPC UA Applications (Clients and Servers) installed should have a *LocalDiscoveryServer* with a *MulticastExtension*.

The *MulticastExtension* incorporates the functionality of the mDNS Responder described in the Multicast DNS (mDNS) specification (see *mDNS*). In addition, the *LocalDiscoveryServer* that supports the *MulticastExtension* supports the *FindServersOnNetwork Service* described in IEC 62541-4.

5.2 Security considerations for Multicast DNS

The Multicast DNS (mDNS) specification is used for various commercial and consumer applications. This provides a benefit in that implementations exist; however, system administrators could choose to disable Multicast DNS operations. For this reason, *Applications* shall not rely on Multicast DNS capabilties.

Multicast DNS operations are insecure because of their nature; therefore, they should be disabled in environments where an attacker could cause problems by impersonating another host. This risk is minimized if OPC UA security is enabled and all *Applications* use *Certificate TrustLists* to control access.

6 Global Discovery Server

6.1 Overview

The *LocalDiscoveryServer* is useful for networks where the host names can be discovered. However, this is typically not the case in large systems with multiple servers on multiple subnets. For this reason, there is a need for an enterprise wide *DiscoveryServer* called a *GlobalDiscoveryServer*. The *GlobalDiscoveryServer* (GDS) is an OPC UA *Server* that allows *Clients* to search for *Servers* in the administrative domain. It provides methods that allow applications to register themselves and to search for other applications.

The essential element of a *GlobalDiscoveryServer* (GDS) is that it can provide the *Certificate* management services defined in Clause 7. These services can simplify *Certificate* management even in medium to small systems; therefore, a GDS can be deployed in smaller systems. Different implementations are expected. Some of them will likely provide a front-end to an existing *DirectoryService* such as LDAP (see Annex E). By standardizing on an OPC UA based interface, OPC UA *Clients* do not need to have knowledge of different *DirectoryServices*.

If an administrator registers a *LocalDiscoveryServer* with the GDS, then the GDS shall periodically update its database by calling *FindServersOnNetwork* or *FindServers* on the LDS. Figure 7 shows the relationship between a GDS and the LDS-ME or LDS.

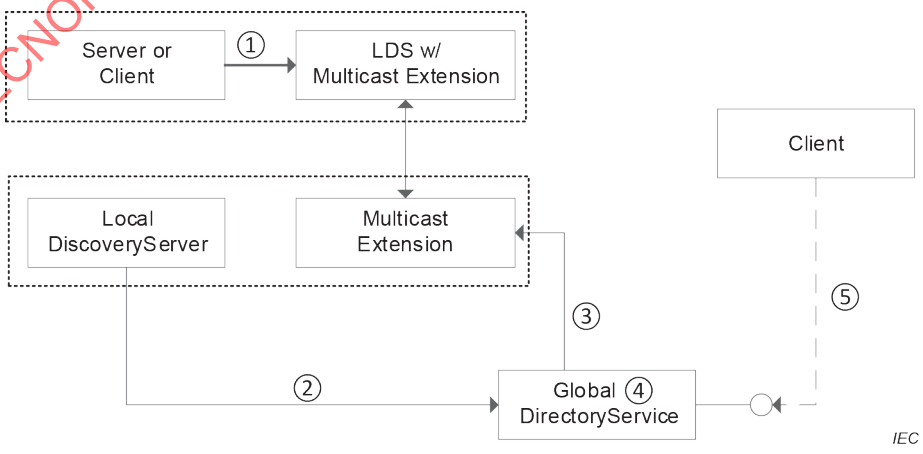


Figure 7 – The relationship between GDS and other components

The steps shown in Figure 7 are:

1) The Server calls *RegisterServer2* on the LDS running on the same machine.
2) The administrator registers LDS-ME installations with the GDS.
3) The GDS calls *FindServersOnNetwork* on the LDS-ME to find all applications on the same *MulticastSubnet*.
4) The GDS creates a record for each application returned by the LDS-ME. These records shall be approved before they are made available to *Clients* of the GDS. This approval can be obtained from an *Administrator or* the GDS can connect to the *Server* and verifies that it has a trusted *Certfiicate.*
5) The *Client* calls *QueryServers Method* on the GDS to discover applications.

The *Information Model* used for registration and discovery is shown in 6.2. Any *Client* shall be able to call the *QueryServers Method* to find applications known to GDS. The complete definitions for each of the types used are described in 7.5.

6.2 Network architectures

6.2.1 Overview

The discovery mechanisms defined in this document are expected to be used in many different network architectures. The following three architectures are illustrated:

- Single *MulticastSubnet*;
- Multiple *MulticastSubnets*;
- No *MulticastSubnet* (or multiple *MulticastSubnets* with exactly one host each).

A *MulticastSubnet* is a network segment where all hosts on the segment can receive multicast packets from the other hosts on the segment. A physical LAN segment is typically a *MulticastSubnet* unless the administrator has specifically disabled multicast communication. In some cases, multiple physical LAN segments can be connected as a Single *MulticastSubnet.*

6.2.2 Single MulticastSubnet

The Single *MulticastSubnet* Architecture is shown in Figure 8.

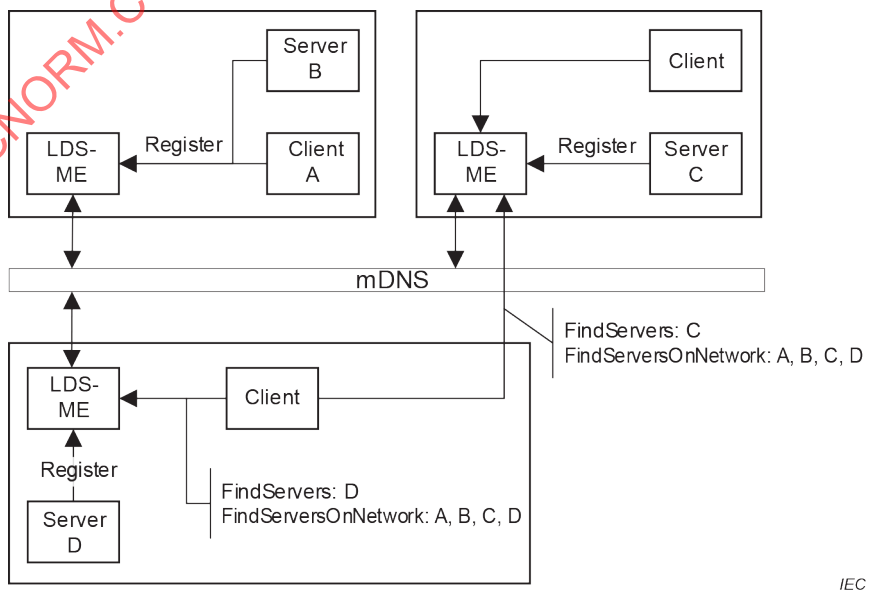


Figure 8 – The Single MulticastSubnet architecture

In this architecture, every host has an LDS-ME and uses mDNS to maintain a cache of the applications on the *MulticastSubnet*. A *Client* can call *FindServersOnNetwork* on any LDS-ME and receive the same set of applications. When a *Client* calls *FindServers*, it only receives the applications running on the same host as the LDS.

6.2.3 Multiple MulticastSubnet

The Multiple *MulticastSubnet* Architecture is shown in Figure 9.

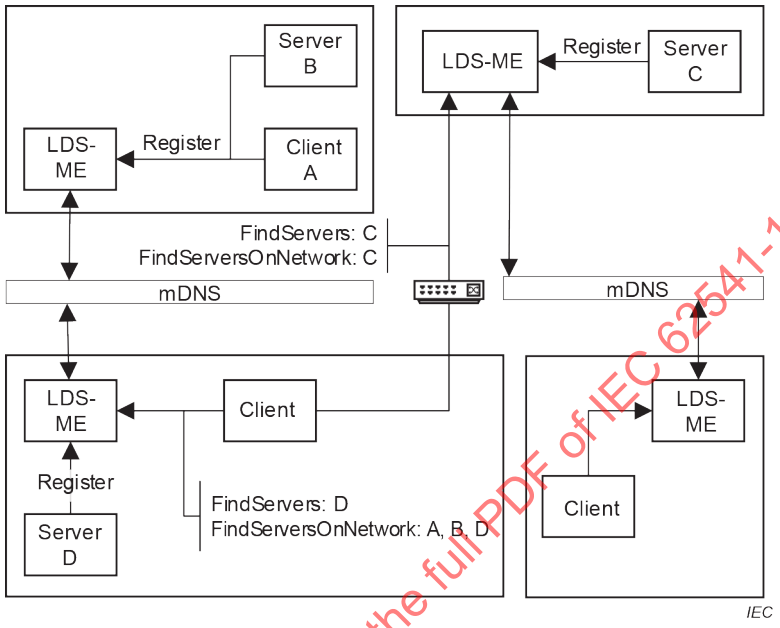


Figure 9 – The Multiple MulticastSubnet architecture

This architecture is the same as the previous architecture, except in this architecture the mDNS messages do not pass through routers connecting the *MulticastSubnets*. This means that a *Client* calling *FindServersOnNetwork* will only receive a list of applications running on the *MulticastSubnets* that the LDS-ME is connected to.

A *Client* that wants to connect to a remote *MulticastSubnet* shall use out of band discovery (i.e. manual entry) of a *HostName* or *DiscoveryUrl*. Once a *Client* finds an LDS-ME on a remote *MulticastSubnet*, it can use *FindServersOnNetwork* to discover all applications on that *MulticastSubnet*.

6.2.4 No MulticastSubnet

The No *MulticastSubnet* Architecture is shown in Figure 10.

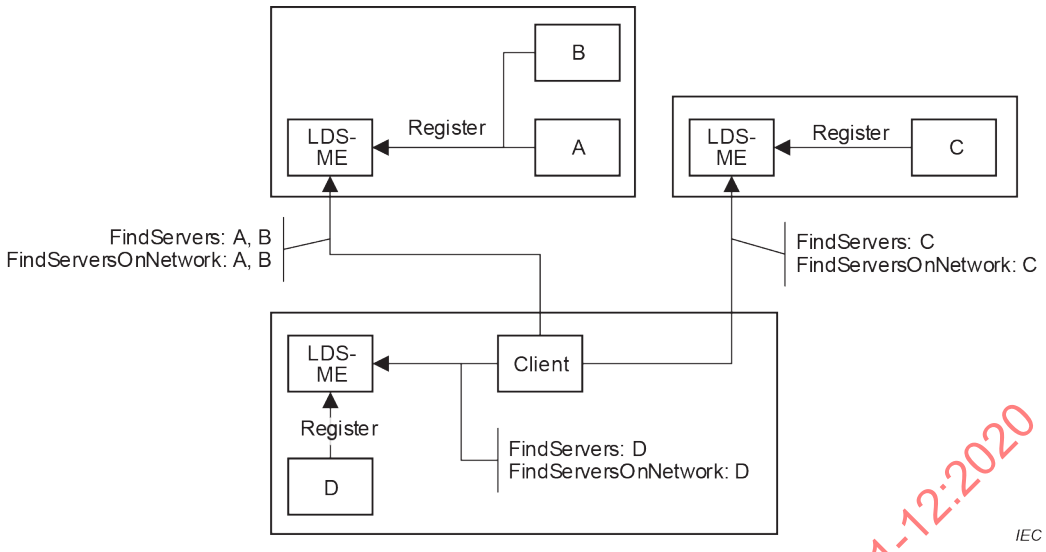


Figure 10 – The No MulticastSubnet architecture

In this architecture, the mDNS is not used at all because the Administrator has disabled multicast at a network level or by turning off multicast capabilities of each LDS-ME.

A *Client* that wants to discover a applications needs to use an out of band mechanism to find the *HostName* and call *FindServers* on the LDS of that host. *FindServersOnNetwork* can also work, but it will never return more than what *FindServers* returns.

6.2.5 Domain Names and MulticastSubnets

The mDNS specification requires that a fully qualified domain name be annouced on the network. If a *Server* is not configured with a fully qualified domain name, then mDNS requires that the "local" top level domain be appended to the domain names. The "local" top level domain indicates that the domain can only be consided to be unique within the subnet where the domain name was used. This means *Clients* need to be aware that URLs received from any LDS-ME other than the one on the *Client's* machine could contain "local" domains that are not reachable or will connect to a different machine with the same domain name that happens to be on the same subnet as the *Client*. It is recommended that *Clients* ignore all URLs with the "local" top level domain unless they are returned from the LDS-ME running on the same machine.

System administrators can eliminate this problem by configuring a normal DNS with the fully qualilfied domain names for all machines that need to be accessed by *Clients* outside the *MulticastSubnet*.

Servers configured with fully qualified domain names should specify the fully qualified domain name in its *ApplicationInstance Certificate*. *Servers* shall not specify domains with the "local" top level domain in their *Certificate*. *Clients* using a URL returned from an LDS-ME shall ignore the "local" top level domain when checking the domain against the *Server Certificate*.

Note that domain name validation is a necessary but not sufficient check against rogue *Servers* or man-in-the-middle attacks when *Server Certificates* do not contain fully qualified domain names. The *Certificate* trust relationship established by administrators is the primary mechanism used to protect against these risks.

6.3 Information Model

6.3.1 Overview

The *GlobalDiscoveryServer Information Model* used for *discovery* is shown in Figure 11. Most of the interactions between the *GlobalDiscoveryServer* and *Application* administrator or the *Client* will be via *Methods* defined on the *Directory* folder.

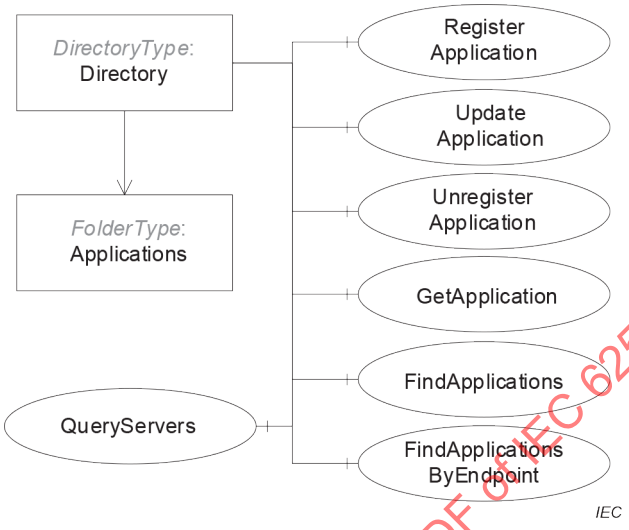


Figure 11 – The Address Space for the GDS

6.3.2 Directory

This *Object* is the root of the *GlobalDiscoveryServer AddressSpace*, and it is the target of an *Organizes* reference from the *Objects* folder defined in IEC 62541-5. It organizes the information that can be accessed into subfolders. The implementation of a GDS can customize and organize the folders in any manner it desires. For example, folders can exist for information models, or for optional services or for various locations in an administrative domain. It is defined in Table 2.

Table 2 – Directory Object definition

Attribute	Value				
BrowseName	Directory				
Namespace	GDS (see 3.3)				
TypeDefinition	DirectoryType defined in 6.3.3.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**

6.3.3 DirectoryType

DirectoryType is the *ObjectType* for the root of the *GlobalDiscoveryServer AddressSpace*. It organizes the information that can be accessed into subfolders. It also provides methods that allow applications to register or find applications. It is defined in Table 3.

Table 3 – DirectoryType definition

Attribute	Value				
BrowseName	DirectoryType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *FolderType* defined in IEC 62541-5.					
Organizes	Object	Applications	-	FolderType	Mandatory
HasComponent	Method	FindApplications	Defined in 6.3.4		Mandatory
HasComponent	Method	RegisterApplication	Defined in 6.3.6		Mandatory
HasComponent	Method	UpdateApplication	Defined in 6.3.7		Mandatory
HasComponent	Method	UnregisterApplication	Defined in 6.3.8		Mandatory
HasComponent	Method	GetApplication	Defined in 6.3.9		Mandatory
HasComponent	Method	QueryApplications	Defined in 6.3.10		Mandatory
HasComponent	Method	QueryServers	Defined in 6.3.11		Mandatory

The *Applications* folder may contain *Objects* representing the *Applications* known to the GDS. These *Objects* may be organized into subfolders as determined by the GDS. Some characteristics for organizing applications are the networks, the physical location, or the supported profiles. The *QueryServers Method* can be used to search for OPC UA *Applications* based on various criteria.

A GDS is not required to expose its *Applications* as browsable *Objects* in its *AddressSpace*, however, each *Application* shall have a unique *NodeId* which can be passed to *Methods* used to administer the GDS.

The *FindApplications Method* returns the *Applications* associated with an *ApplicationUri*. It can be called by any *Client* application.

The *RegisterApplication Method* is used to add a new *Application* to the GDS. It requires administrative privileges.

The *UpdateApplication Method* is used to update an existing *Application* in the GDS. It requires administrative privileges.

The *UnregisterApplication Method* is used to remove an *Application* from the GDS. It requires administrative privileges.

The *QueryApplications Method* is used to find *Client* or *Server* applications that meet the criteria provided. This *Method* replaces the *QueryServers Method.*

The *QueryServers Method* is used to find *Servers* that meet the criteria specified. It can be called by any *Client* application. This *Method* has been replaced by the *QueryApplications Method*.

6.3.4 FindApplications

FindApplications is used to find the *ApplicationId* for an OPC UA *Application* known to the GDS. In normal situations, the list of records returned will not have more than one entry; however, system configuration errors can create situations where the GDS has multiple entries for a single *ApplicationUri*. If this happens, a human will likely have to look at records to determine which record is the true match for the *ApplicationUri*.

If the returned array is null or zero length then the GDS does not have an entry for the *ApplicationUri*.

Signature

```
FindApplications(
      [in]  String applicationUri
      [out] ApplicationRecordDataType[] applications
      );
```

Argument	Description
applicationUri	The *ApplicationUri* that identifies the *Application* of interest.
applications	A list of application records that match the *ApplicationUri*. The ApplicationRecordDataType is defined in 6.3.5.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 4 specifies the *AddressSpace* representation for the *FindApplications Method*.

Table 4 – FindApplications Method AddressSpace definition

Attribute	Value				
BrowseName	FindApplications				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.3.5 ApplicationRecordDataType

This type defines a DataType which represents a record in the GDS (see Table 5).

Table 5 – ApplicationRecordDataType definition

Name	Type	Value
applicationId	NodeId	The unique identifier assigned by the GDS to the record. This *NodeId* may be passed to other *Methods*.
applicationUri	String	The URI for the *Application* associated with the record.
applicationType	ApplicationType	The type of application. This type is defined in IEC 62541-4.
applicationNames	LocalizedText[]	One or more localized names for the application. The first element is the default *ApplicationName* for the application when a non-localized name is needed.
productUri	String	A globally unique URI for the product associated with the application. This URI is assigned by the vendor of the application.
discoveryUrls	String[]	The list of discovery URLs for an application. The first element is the default if a *Client* needs to choose one URL. The first HTTPS URL specifies the domain used as the Common Name of HTTPS *Certificates*. If the *ApplicationType* is *Client* then all of the URLs shall have the 'inv+' prefix which indicates they support reverse connect.
serverCapability Identifiers	String[]	The list of server capability identifiers for the application. The allowed values are defined in Annex D.

6.3.6 RegisterApplication

RegisterApplication is used to register a new *Application* Instance with a *GlobalDiscoveryServer*.

This *Method* shall only be invoked by authorized users.

Servers that support transparent redundancy shall register as a single application and pass the *DiscoveryUrls* for all available instances and/or network paths.

RegisterApplication will create duplicate records if the *ApplicationUri* already exists since misconfiguration of applications can result in different applications having the same *ApplicationUri*. Before calling this *Method* the *Client* shall call *FindApplications* to check if a record for the application it is using already exists. If records are found that appear to belong to different applications (e.g. the *DiscoveryUrls* are different) then the *Client* shall report a warning before continuing.

If registration was successful and auditing is supported, the GDS shall generate the *ApplicationRegistrationChangedAuditEventType* (see 6.3.12).

Signature

```
RegisterApplication(
    [in]  ApplicationRecordDataType application
     [out] NodeId applicationId
);
```

Argument	Description
application	The application that is to be registered with the *GlobalDiscoveryServer*.
applicationId	A unique identifier for the registered *Application*. This identifier is persistent and is used in other *Methods* used to administer applications.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The application or one of the fields of the application record is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 6 specifies the *AddressSpace* representation for the *RegisterApplication Method*.

Table 6 – RegisterApplication Method AddressSpace definition

Attribute	Value				
BrowseName	RegisterApplication				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.3.7 UpdateApplication

UpdateApplication is used to update an existing *Application* in a *GlobalDiscoveryServer*.

This *Method* shall only be invoked by authorized users.

If the update was successful and auditing is supported, the GDS shall generate the *ApplicationRegistrationChangedAuditEventType* (see 6.3.12).

Signature

```
UpdateApplication(
      [in]  ApplicationRecordDataType application
    );
```

Argument	Description
application	The application that is to be updated in the GDS database.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The applicationId is not known to the GDS.
Bad_InvalidArgument	The application or one of the fields of the application record is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 7 specifies the *AddressSpace* representation for the *UpdateApplication Method*.

Table 7 – UpdateApplication Method AddressSpace definition

Attribute	Value				
BrowseName	UpdateApplication				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

6.3.8 UnregisterApplication

UnregisterApplication is used to remove an *Application* from a *GlobalDiscoveryServer*.

This *Method* shall only be invoked by authorized users.

A *Server Application* that is unregistered may be automatically added again if the GDS is configured to populate itself by calling *FindServersOnNetwork* and the *Server Application* is still registering with its local LDS.

If un-registration was successful and auditing is supported, the GDS shall generate the *ApplicationRegistrationChangedAuditEventType* (see 6.3.12).

Signature

```
UnregisterApplication(
      [in] NodeId applicationId
     );
```

Argument	Description
applicationId	The identifier assigned by the GDS to the *Application*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *ApplicationId* is not known to the GDS.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 8 specifies the *AddressSpace* representation for the *UnregisterApplication Method*.

Table 8 – UnregisterApplication Method AddressSpace definition

Attribute	Value				
BrowseName	UnregisterApplication				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

6.3.9 GetApplication

GetApplication is used to find an OPC UA *Application* known to the GDS.

Signature

```
GetApplication(
      [in]  NodeId applicationId
      [out] ApplicationRecordDataType application
      );
```

Argument	Description
applicationId	The *ApplicationId* that identifies the *Application* of interest.
application	The application record that matches the *ApplicationId*. The ApplicationRecordDataType is defined in 6.3.5.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The no record found for the specified *ApplicationId*.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 9 specifies the *AddressSpace* representation for the *GetApplication Method*.

Table 9 – GetApplication Method AddressSpace definition

Attribute	Value				
BrowseName	GetApplication				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.3.10 QueryApplications

QueryApplications is used to find *Client* or *Server* applications that meet the specified filters. The only Clients returns are those that support the reverse connection capability described in IEC 62541-6.

QueryApplications returns *ApplicationDescriptions* instead of the *ServerOnNetwork Structures* returned by *QueryServers*. This is more useful to some *Clients* because it matches the return type of *FindServers*.

Any *Client* is able to call this *Method*, however, the set of results returned may be restricted based on the *Client's* user credentials.

The applications returned shall pass all of the filters provided (i.e. the filters are combined in an AND operation). The *capabilities* parameter is an array and an application will pass this filter if it supports all of the specified capabilities.

Each time the GDS creates or updates an application record it shall assign a monotonically increasing identifier to the record. This allows *Clients* to request records in batches by specifying the identifier for the last record received in the last call to *QueryApplications*. To support this, the GDS shall return records in order starting from the lowest record identifier. The GDS shall also return the last time the counter was reset. If a *Client* detects that this time is more recent than the last time the *Client* called the *Method* it shall call the *Method* again with a *startingRecordId* of 0.

Signature

```
QueryApplications(
      [in]  UInt32 startingRecordId
      [in]  UInt32 maxRecordsToReturn
      [in]  String applicationName
      [in]  String applicationUri
      [in]  UInt32 applicationType
      [in]  String productUri
      [in]  String[] capabilities
      [out] DateTime lastCounterResetTime
      [out] UInt32 nextRecordId
      [out] ApplicationDescription[] applications
    );
```

Argument	Description
INPUTS	
startingRecordId	Only records with an identifier greater than this number will be returned. Specify 0 to start with the first record in the database.
maxRecordsToReturn	The maximum number of records to return in the response. 0 indicates that there is no limit.
applicationName	The *ApplicationName* of the applications to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified. The filter is only applied to the default *ApplicationName*.
applicationUri	The *ApplicationUri* of the applications to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified.
applicationType	A mask indicating what types of applications are returned. The mask values are: 0x1 – Servers; 0x2 – Clients;: If the mask is 0 then all applications are returned.
productUri	The *ProductUri* of the applications to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified.
capabilities	The capabilities supported by the applications returned. The applications returned shall support all of the capabilities specified. If no capabilities are provided this filter is not used.
OUTPUTS	
lastCounterResetTime	The last time the counters were reset.
nextRecordId	The identifier of the next record. It is passed as the *startingRecordId* in subsequent calls to *QueryApplications* to fetch the next batch of records. It is 0 if there are no more records to return.
applications	A list of *Applications* that meet the criteria. The *ApplicationDescription* structure is defined in IEC 62541-4.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 10 specifies the *AddressSpace* representation for the *QueryApplications Method*.

Table 10 – QueryApplications Method AddressSpace definition

Attribute	Value				
BrowseName	QueryApplications				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.3.11 QueryServers (deprecated)

QueryServers is used to find *Server* applications that meet the specified filters.

Any *Client* is able to call this *Method*, however, the set of results returned may be restricted based on the *Client's* user credentials.

The applications returned shall pass all of the filters provided (i.e. the filters are combined in an AND operation). The *serverCapabilities* parameter is an array and an application will pass this filter if it supports all of the specified capabilities.

Each time the GDS creates or updates an application record it shall assign a monotonically increasing identifier to the record. This allows *Clients* to request records in batches by specifying the identifier for the last record received in the last call to *QueryServers*. To support this, the GDS shall return records in order starting from the lowest record identifier. The GDS shall also return the last time the counter was reset. If a *Client* detects that this time is more recent than the last time the *Client* called the *Method* it shall call the *Method* again with a *startingRecordId* of 0.

Signature

```
QueryServers(
      [in]  UInt32 startingRecordId
      [in]  UInt32 maxRecordsToReturn
      [in]  String applicationName
      [in]  String applicationUri
      [in]  String productUri
      [in]  String[] serverCapabilities
      [out] DateTime lastCounterResetTime
      [out] ServerOnNetwork[] servers
    );
```

Argument	Description
INPUTS	
startingRecordId	Only records with an identifier greater than this number will be returned. Specify 0 to start with the first record in the database.
maxRecordsToReturn	The maximum number of records to return in the response. 0 indicates that there is no limit.
applicationName	The *ApplicationName* of the *Applications* to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified. The filter is only applied to the default *ApplicationName*.
applicationUri	The *ApplicationUri* of the *Servers* to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified.
productUri	The *ProductUri* of the *Servers* to return. Supports the syntax used by the LIKE *FilterOperator* described in IEC 62541-4. Not used if an empty string is specified.
serverCapabilities	The applications returned shall support all of the server capabilities specified. If no server capabilities are provided this filter is not used.
OUTPUTS	
lastCounterResetTime	The last time the counters were reset.
servers	A list of *Servers* which meet the criteria. The *ServerOnNetwork* structure is defined in IEC 62541-4.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 11 specifies the *AddressSpace* representation for the *QueryServers Method*.

Table 11 – QueryServers Method AddressSpace definition

Attribute	Value				
BrowseName	QueryServers				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

6.3.12 ApplicationRegistrationChangedAuditEventType

This event is raised when the *RegisterApplication*, *UpdateApplication* or *UnregisterApplication Methods* are called.

Its representation in the *AddressSpace* is formally defined in Table 12.

Table 12 – ApplicationRegistrationChangedAuditEventType definition

Attribute	Value				
BrowseName	ApplicationRegistrationChangedAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantics are defined in IEC 62541-5.

7 Certificate management overview

7.1 Overview

Certificate management functions comprise the management and distribution of certificates and *Trust Lists* for OPC UA Applications. An application that provides the certificate management functions is called *CertificateManager*. GDS and *CertificateManager* will typically be combined in one application. The basic concepts regarding *Certificate* management are described in IEC TR 62541-2.

There are two primary models for *Certificate* management: pull and push management. In pull management, the application acts as a *Client* and uses the *Methods* on the *CertificateManager* to request and update *Certificates* and *Trust Lists*. The application is responsible for ensuring the *Certificates* and *Trust Lists* are kept up to date. In push management, the application acts as a *Server* and exposes *Methods* which the *CertificateManager* can call to update the *Certificates* and *Trust Lists* as required.

The GDS is intended to work in conjunction with different Certificate Management services such as Active Directory. The GDS provides a standard OPC UA based information model that all OPC UA applications can support without needing to know the specifics of a particular Certificate Management system.

The *CertificateManager* should support the following features:

- provisioning (first time setup for a device/application);
- renewal (renewing expired or compromised certificates);
- *Trust List* update (updating the *Trust Lists* including the *Revocation Lists*);
- revocation (removing a device/application from the system).

Although it is generally assumed that *Client* applications will use the Pull model and *Server* applications will use the Push model, this is not required.

During provisioning, the *CertificateManager* shall be able to operate in a mode where any *Client* is allowed to connect securely with any valid *Certificate* and user credentials are used to determine the rights a *Client* has; this eliminates the need to configure *Trust Lists* before connecting to the *CertificateManager* for provisioning.

Application vendors may decide to build the interaction with the *CertificateManager* as a separate component, e.g. as part of an administration application with access to the OPC UA configuration of this *Application*. This is transparent for the *CertificateManager* and will not be considered further in the rest of this clause.

This document does not define how to administer a *CertificateManager* but a *CertificateManager* shall provide an integrated system that includes both push and pull management.

7.2 Pull Management

Pull Management is performed by using the *CertificateManager* information model – in particular the Methods - defined in 7.6. The interactions between *Application* and *CertificateManager* during Pull Management are illustrated in Figure 12.

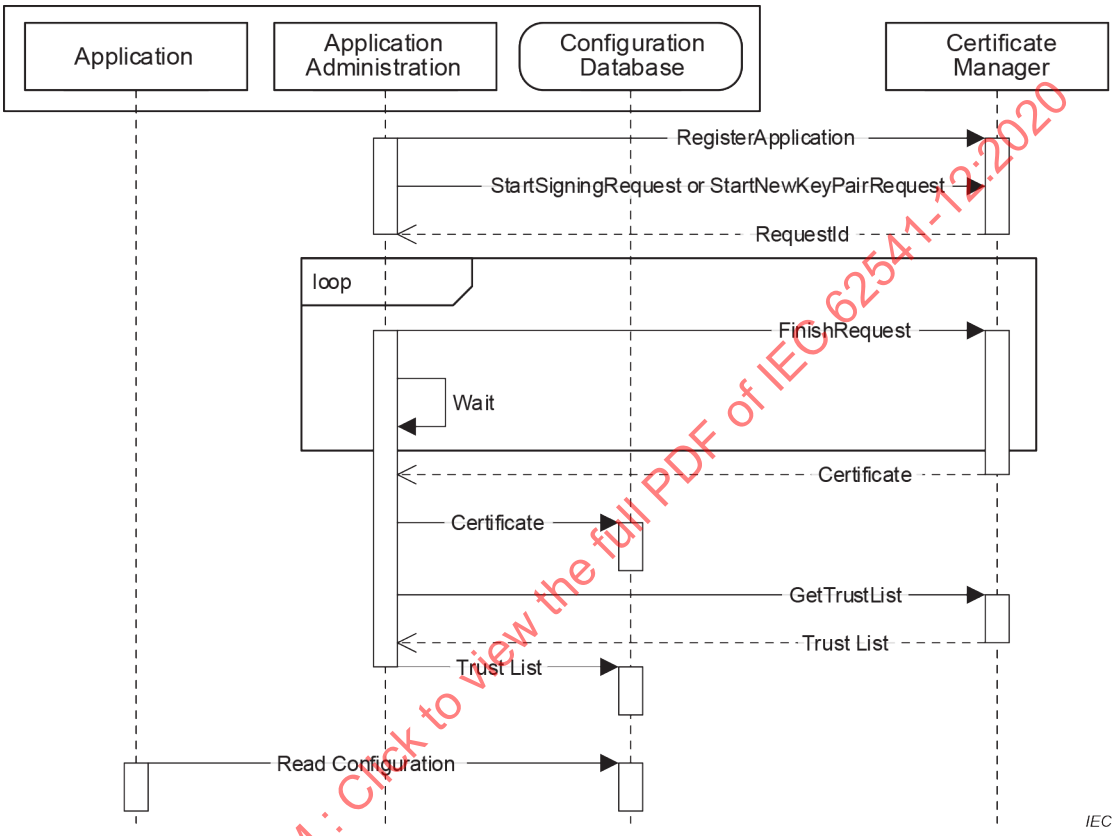


Figure 12 – The Pull Certificate management model

The Application Administration component may be part of the Application or a standalone utility that understands how the Application persists its configuration information in its Configuration Database.

A similar process is used to renew certificates or to periodically update *Trust List*.

Security in Pull management requires an encrypted channel and the use of *Administrator* credentials for the *CertificateManager* that ensure only authorized users can register new *Applications* and request an initial new *Certificate*. Once an *Application* has a *Certificate* it can use this *Certificate* to renew the *Certificate* or to update *Trust Lists* and *Revocation* lists. It is important that a *CertificateManager* does not provide certificate renewals except to the applications that already own the prior certificate.

7.3 Push management

Push management is targeted at *Server* applications and relies on *Methods* defined in 7.7 to get a *Certificate Request* which can be passed onto the *CertificateManager.* After the *CertificateManager* signs the *Certificate*, the new *Certificate* is pushed to the *Server* with the *UpdateCertificate Method*.

The interactions between a *Server Application* and *CertificateManager* during Push Management are illustrated in Figure 13.

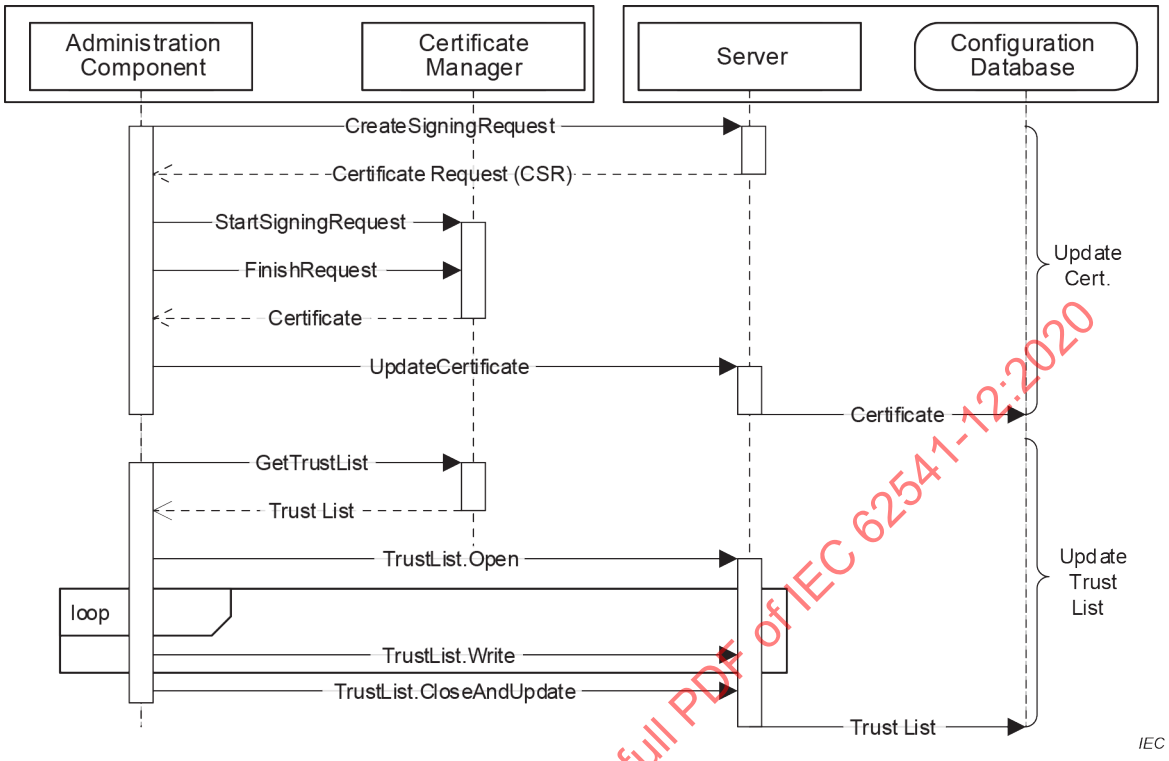


Figure 13 – The Push Certificate management model

The Administration Component may be part of the *CertificateManager* or a standalone utility that uses OPC UA to communicate with the *CertificateManager* (see 7.2 for a more complete description of the interactions required for this use case). The Configuration Database is used by the *Server* to persist its configuration information. The *RegisterApplication Method* (or internal equivalent) is assumed to have been called before the sequence in the diagram starts.

A similar process is used to renew certificates or to periodically update *Trust List*.

Security when using the Push Management Model requires an encrypted channel and the use of Administrator credentials for the *Server* that ensure only authorized users can update *Certificates* or *Trust Lists*. In addition, separate *Administrator* credentials are required for the *CertificateManager* that ensure only authorized users can register new *Servers* and request new *Certificates*.

7.4 Provisioning

Provisioning is the initial installation of an OPC UA *Server* or *Client* into a system in which a GDS is available and managing all certificates. For applications using a *Client* interface, provisioning can be accomplished using a Pull Model. Applications using the *Server* interface can be provisioned using the Push Model.

OPC UA *Servers* will typically auto-generate a self-signed *Certificate* when they first start. They may also have a pre-configured *Trust List* with *Applications* that are allowed to provision the *Server*. For example, a device vendor may use a CA that is used to issue *Certificates* to *Applications* used by their field technicians.

For embedded devices, the *Server* should allow any *Client* that provides the proper *Administrator* credentials to create the secure connection needed for provisioning using push management. Once the device has been given its initial *Trust List*, the *Server* should then restrict access to those *Clients* with *Certificates* in the *Trust List*. A vendor-specific process for provisioning is required if a device does not allow any *Client* to connect securely for provisioning.

See Clause G.1 for more specific examples of how to provision an application.

7.5 Common Information Model

7.5.1 Overview

The common information model defines types that are used in both the Push and the Pull Model.

7.5.2 TrustListType

This type defines a *FileType* that can be used to access a *Trust List*. The *TrustListType* is formally defined in Table 13.

The *CertificateManager* uses this type to implement the Pull Model.

Servers use this type when implementing the Push Model.

An instance of a *TrustListType* shall restrict access to appropriate users or applications. This may be a *CertificateManager* administrative user that can change the contents of a *Trust List*, it may be an Administrative user that is reading a *Trust List* to deploy to an Application host, or it may be an Application that can only access the Trust List assigned to it.

The *Trust List* file is a UA Binary encoded stream containing an instance of *TrustListDataType* (see 7.5.7).

The *Open Method* shall not support modes other than Read (0x01) and the Write + EraseExisting (0x06).

When a *Client* opens the file for writing the *Server* will not actually update the *Trust List* until the *CloseAndUpdate Method* is called. Simply calling *Close* will discard the updates. The bit masks in *TrustListDataType* structure allow the *Client* to only update part of the *Trust List*.

When the *CloseAndUpdate Method* is called the *Server* will validate all new *Certificates* and *CRLs*. If this validation fails the *Trust List* is not updated and the *Server* returns the appropriate *Certificate* error code (see IEC 62541-4).

Table 13 – TrustListType definition

Attribute	Value				
BrowseName	TrustListType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *FileType* defined in IEC 62541-5.					
HasProperty	Variable	LastUpdateTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	UpdateFrequency	Duration	PropertyType	Optional
HasComponent	Method	OpenWithMasks	Defined in 7.5.3		Optional
HasComponent	Method	CloseAndUpdate	Defined in 7.5.4		Optional
HasComponent	Method	AddCertificate	Defined in 7.5.5		Optional
HasComponent	Method	RemoveCertificate	Defined in 7.5.6		Optional

The *LastUpdateTime* indicates when the *Trust List* was last updated via *Trust List Object Methods*. This can be used to determine if a device has an up to date *Trust List* or to detect unexpected modifications. Out of band changes are not necessarily reported by this value.

The *UpdateFrequency Property* specifies how often the *Trust List* needs to be checked for changes. When the *CertificateManager* specifies this value, all *Clients* that read a copy of the *Trust List* should connect to the *CertificateManager* and check for updates to the *Trust List* within 2 times the *UpdateFrequency*. If the *Trust List Object* is contained within a *ServerConfiguration Object* then this value specifies how frequently the *Server* expects the *Trust List* to be updated.

If auditing is supported, the CertificateManager shall generate the *TrustListUpdatedAuditEventType* (see 7.5.18) if the *CloseAndUpdate*, *AddCertificate* or *RemoveCertificate Methods* are called.

7.5.3 OpenWithMasks

The *OpenWithMasks Method* allows a *Client* to read only the portion of the *Trust List.*

This *Method* can only be used to read the *Trust List*.

Signature

```
OpenWithMasks(
      [in]  UInt32 masks
      [out] UInt32 fileHandle
     );
```

Argument	Description
masks	The parts of the *Trust List* that are include in the file to read. The masks are defined in 7.5.8.
fileHandle	The handle of the newly opened file.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 14 specifies the *AddressSpace* representation for the *OpenWithMasks Method*.

Table 14 – OpenWithMasks Method AddressSpace definition

Attribute	Value				
BrowseName	OpenWithMasks				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.5.4 CloseAndUpdate

The *CloseAndUpdate Method* closes the file and applies the changes to the *Trust List*. It can only be called if the file was opened for writing. If the *Close Method* is called any cached data is discarded and the *Trust List* is not changed.

The *Server* shall verify that every *Certificate* in the new *Trust List* is valid in accordance with the mandatory rules defined in IEC 62541-4. If an invalid *Certificate* is found, the *Server* shall return an error and shall not update the *Trust List*. If only part of the *Trust List* is being updated, the *Server* creates a temporary *Trust List* that includes the existing *Trust List* plus any updates and validates the temporary *Trust List*.

If the file cannot be processed this *Method* still closes the file and discards the data before returning an error. This *Method* is required if the *Server* supports updates to the *Trust List*.

The structure uploaded includes a mask (see 7.5.8) that specifies which fields are updated. If a bit is not set, then the associated field is not changed.

Signature

```
CloseAndUpdate(
      [in]  UInt32 fileHandle
      [out] Boolean applyChangesRequired
     );
```

Argument	Description
fileHandle	The handle of the previously opened file.
applyChangesRequired	A flag indicating whether the *ApplyChanges* Method (see 7.7.5) shall be called before the new *Trust List* will be used by the *Server*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_CertificateInvalid	The Server could not validate all Certificates in the Trust List. The DiagnosticInfo shall specify which Certificate(s) are invalid and the specific error.

Table 15 specifies the *AddressSpace* representation for the *CloseAndUpdate Method*.

Table 15 – CloseAndUpdate Method AddressSpace definition

Attribute	Value				
BrowseName	CloseAndUpdate				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.5.5 AddCertificate

The *AddCertificate Method* allows a *Client* to add a single *Certificate* to the *Trust List*. The *Server* shall verify that the *Certificate* is valid according to the rules defined in IEC 62541-4. If an invalid *Certificate* is found the *Server* shall return an error and shall not update the *Trust List*.

This *Method* will return a validation error if the *Certificate* is issued by a CA and the *Certificate* for the issuer is not in the *Trust List*.

This *Method* cannot provide CRLs, so issuer Certificates cannot be added with this *Method*. Instead, CA *Certificates* and their CRLs shall be managed with the *Write Method* on the containing *TrustList Object.*

This *Method* cannot be called if the containing *TrustList Object* is open.

```
AddCertificate(
      [in] ByteString certificate
      [in] Boolean isTrustedCertificate
    );
```

Argument	Description
Certificate	The DER encoded Certificate to add.
isTrustedCertificate	If TRUE the Certificate is added to the Trusted Certificates List. If FALSE the Certificate is added to the Issuer Certificates List.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_CertificateInvalid	The certificate to add is invalid.
Bad_InvalidState	The object is opened.

Table 16 specifies the *AddressSpace* representation for the *AddCertificate Method*.

Table 16 – AddCertificate Method AddressSpace definition

Attribute	Value				
BrowseName	AddCertificate				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

7.5.6 RemoveCertificate

The *RemoveCertificate Method* allows a *Client* to remove a single *Certificate* from the *Trust List*. It returns *Bad_InvalidArgument* if the thumbprint does not match a Certificate in the *Trust List*.

If the *Certificate* is a CA *Certificate* with associated CRLs then all CRLs are removed as well.

This method cannot be called if the file object is open.

```
RemoveCertificate(
      [in] String thumbprint
      [in] Boolean isTrustedCertificate
     );
```

Argument	Description
Thumbprint	The SHA1 hash of the *Certificate* to remove.
isTrustedCertificate	If TRUE the Certificate is removed from the Trusted Certificates List. If FALSE the Certificate is removed from the Issuer Certificates List.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_InvalidArgument	The certificate to remove was not found.
Bad_InvalidState	The object is opened.

Table 17 specifies the *AddressSpace* representation for the *RemoveCertificate Method*.

Table 17 – RemoveCertificate Method AddressSpace definition

Attribute	Value				
BrowseName	RemoveCertificate				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

7.5.7 TrustListDataType

This type defines a DataType that stores the Trust List of a *Server.* Its values are defined in Table 18.

Table 18 – TrustListDataType definition

Name	Type	Value
TrustListDataType	structure	
specifiedLists	TrustListMasks	A bit mask that indicates which lists contain information. The *TrustListMasks* enumeration in 7.5.8 defines the allowed values.
trustedCertificates	ByteString[]	The list of *Application* and CA *Certificates* which are trusted.
trustedCrls	ByteString[]	The CRLs for the *Certificates* in the *trustedCertificates* list.
issuerCertificates	ByteString[]	The list of CA *Certificates* that are necessary to validate *Certificates*.
issuerCrls	ByteString[]	The CRLs for the CA *Certificates* in the *issuerCertificates* list.

7.5.8 TrustListMasks

This is a DataType that defines the values used for the SpecifiedLists field in the *TrustListDataType*. Its values are defined in Table 19.

Table 19 – TrustListMasks values

Value	Value
None_0	No fields are provided.
TrustedCertificates_1	The TrustedCertificates are provided.
TrustedCrls_2	The TrustedCrls are provided.
IssuerCertificates_4	The IssuerCertificates are provided.
IssuerCrls_8	The IssuerCrls are provided.
All_15	All fields are provided.

7.5.9 TrustListOutOfDateAlarmType

This *SystemOffNormalAlarmType* is raised by the *Server* when the *UpdateFrequency* elapses and the *Trust List* has not been updated. This alarm automatically returns to normal when the *Trust List* is updated. This type is defined in Table 20.

Table 20 – TrustListOutOfDateAlarmType definition

Attribute	Value				
BrowseName	TrustListOutOfDateAlarmType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the SystemOffNormalAlarmType defined in IEC 62541-9.					
HasProperty	Variable	TrustListId	NodeId	PropertyType	Mandatory
HasProperty	Variable	LastUpdateTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	UpdateFrequency	Duration	PropertyType	Mandatory

TrustListId Property specifies the *NodeId* of the out of date *Trust List Object*.

LastUpdateTime Property specifies when the *Trust List* was last updated.

UpdateFrequency Property specifies how frequently the *Trust List* needs to be updated.

7.5.10 CertificateGroupType

This type is used for *Objects* that represent *Certificate Groups* in the *AddressSpace*. A *Certificate Group* is a context that contains a *Trust List* and one or more *Certificates* that can be assigned to an *Application*. This type exists to allow an *Application* that has multiple *Trust Lists* and/or *Application Certificates* to express them in its *AddressSpace*. This type is defined in Table 21.

Table 21 – CertificateGroupType definition

Attribute	Value				
BrowseName	CertificateGroupType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasComponent	Object	TrustList	-	TrustListType	Mandatory
HasProperty	Variable	CertificateTypes	NodeId[]	PropertyType	Mandatory
HasComponent	Object	CertificateExpired		CertificateExpirationAlarmType	Optional
HasComponent	Object	TrustListOutOfDate		TrustListOutOfDateAlarmType	Optional

The *TrustList Object* is the *Trust List* associated with the *Certificate Group*.

The *CertificateTypes Property* specifies the *NodeIds* of the *CertificateTypes* that may be assigned to *Applications* that belong to the *Certificate Group*. For example, a *Certificate Group* with the *NodeId* of *RsaMinApplicationCertificateType* (see 7.5.15) and the *NodeId RsaSha256ApplicationCertificate* (see 7.5.16) specified allows an *Application* to have one *Application Instance Certificates* for each type. Abstract base types may be used in this value and indicate that any subtype is allowed. If this list is empty, then the *Certificate Group* does not allow *Certificates* to be assigned to *Applications* (i.e. the *Certificate Group* exists to allow the associated *Trust List* to be read or updated). All *CertificateTypes* for a given *Certificate Group* shall be subtypes of a single common type which shall be either *ApplicationCertificateType* or *HttpsCertificateType*.

The *CertificateExpired Object* is an *Alarm* which is raised when the *Certificate* associated with the *CertificateGroup* is about to expire. The *CertificateExpirationAlarmType* is defined in IEC 62541-9.

The *TrustListOutOfDate Object* is an Alarm which is raised when the *Trust List* has not been updated within the period specified by the *UpdateFrequency* (see 7.5.2). The *TrustListOutOfDateAlarmType* is defined in 7.5.9.

7.5.11 CertificateType

This type is an abstract base type for types that describe the purpose of a *Certificate*. This type is defined in Table 22.

Table 22 – CertificateType definition

Attribute	Value				
BrowseName	CertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5					
HasSubtype	ObjectType	ApplicationCertificateType	Defined in 7.5.12		
HasSubtype	ObjectType	HttpsCertificateType	Defined in 7.5.13		
HasSubtype	ObjectType	UserCredentialCertificateType	Defined in 7.5.14		

7.5.12 ApplicationCertificateType

This type is an abstract base type for types that describe the purpose of an *ApplicationInstanceCertificate*. This type is defined in Table 23.

Table 23 – ApplicationCertificateType definition

Attribute	Value				
BrowseName	ApplicationCertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *CertificateType* defined in 7.5.11					
HasSubtype	ObjectType	RsaMinApplicationCertificateType	Defined in 7.5.15		
HasSubtype	ObjectType	RsaSha256ApplicationCertificateType	Defined in 7.5.16		

7.5.13 HttpsCertificateType

This type is used to describe Certificates that are intended for use as HTTPS *Certificates*. This type is defined in Table 24.

Table 24 – HttpsCertificateType definition

Attribute	Value				
BrowseName	HttpsCertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *CertificateType* defined in 7.5.11.					

7.5.14 UserCredentialCertificateType

This type is used to describe Certificates that are intended for use as user credentials. This type is defined in Table 25.

Table 25 – UserCredentialCertificateType definition

Attribute	Value				
BrowseName	UserCredentialCertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *CertificateType* defined in 7.5.11.					

7.5.15 RsaMinApplicationCertificateType

This type is used to describe *Certificates* intended for use as an *ApplicationInstanceCertificate*. They shall have an RSA key size of 1 024 bits or 2 048 bits. All *Applications* which support the *Basic128Rsa15* and *Basic256* profiles (see IEC 62541-7) shall have a *Certificate* of this type. This type is defined in Table 26.

Table 26 – RsaMinApplicationCertificateType definition

Attribute	Value				
BrowseName	RsaMinApplicationCertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *ApplicationCertificateType* defined in 7.5.12.					

7.5.16 RsaSha256ApplicationCertificateType

This type is used to describe *Certificates* intended for use as an *ApplicationInstanceCertificate*. They shall have an RSA key size of 2 048 bits, 3 072 bits or 4 096 bits. All *Applications* that support the *Basic256Sha256* profile (see IEC 62541-7) shall have a *Certificate* of this type. This type is defined in Table 27.

Table 27 – RsaSha256ApplicationCertificateType definition

Attribute	Value				
BrowseName	RsaSha256ApplicationCertificateType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *ApplicationCertificateType* defined in 7.5.12					

7.5.17 CertificateGroupFolderType

This type is used for *Folders* that organize *Certificate Groups* in the *AddressSpace*. This type is defined in Table 28.

Table 28 – CertificateGroupFolderType definition

Attribute	Value				
BrowseName	CertificateGroupFolderType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *FolderType* defined in IEC 62541-5.					
Organizes	Object	DefaultApplicationGroup		CertificateGroupType	Mandatory
Organizes	Object	DefaultHttpsGroup		CertificateGroupType	Optional
Organizes	Object	DefaultUserTokenGroup		CertificateGroupType	Optional
Organizes	Object	<AdditionalGroup>		CertificateGroupType	Optional Placeholder

The *DefaultApplicationGroup Object* represents the default *Certificate Group* for *Applications*. It is used to access the default *Application Trust List* and to define the *CertificateTypes* allowed for the *ApplicationInstanceCertificate*. This *Object* shall specify the *ApplicationCertificateType NodeId* (see 7.5.12) as a single entry in the *CertificateTypes* list or it shall specify one or more subtypes of *ApplicationCertificateType*.

The *DefaultHttpsGroup Object* represents the default *Certificate Group* for HTTPS communication. It is used to access the default HTTPS *Trust List* and to define the *CertificateTypes* allowed for the *HTTPS Certificate*. This *Object* shall specify the *HttpsCertificateType NodeId* (see 7.5.13) as a single entry in the *CertificateTypes* list or it shall specify one or more subtypes of *HttpsCertificateType*.

This *DefaultUserTokenGroup Object* represents the default *Certificate Group* for validating user credentials. It is used to access the default user credential *Trust List* and to define the *CertificateTypes* allowed for user credentials *Certificate*. This *Object* shall leave *CertificateTypes* list empty.

7.5.18 TrustListUpdatedAuditEventType

This event is raised when a *Trust List* is changed.

This is the result of a *CloseAndUpdate Method* on a *TrustListType Object* being called.

It shall also be raised when the *AddCertificate* or *RemoveCertificate* Method causes an update to the *Trust List*.

Its representation in the *AddressSpace* is formally defined in Table 29.

Table 29 – TrustListUpdatedAuditEventType definition

Attribute	Value				
BrowseName	TrustListUpdatedAuditEventType				
Namespace	CORE (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

7.6 Information Model for Pull Certificate Management

7.6.1 Overview

The *GlobalDiscoveryServer AddressSpace* used for *Certificate* management is shown in Figure 14. Most of the interactions between the *GlobalDiscoveryServer* and *Application* administrator or the *Client* will be via *Methods* defined on the *Directory* folder.

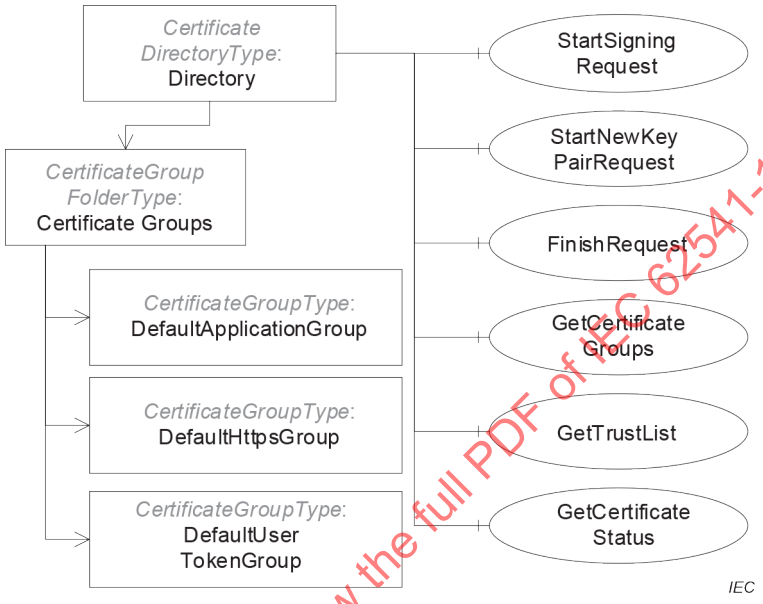


Figure 14 – The Certificate Management AddressSpace for the GlobalDiscoveryServer

7.6.2 CertificateDirectoryType

This *ObjectType* is the *TypeDefinition* for the root of the *CertificateManager AddressSpace*. It provides additional *Methods* for *Certificate* management which are shown in Table 30.

Table 30 – CertificateDirectoryType ObjectType definition

Attribute	Value				
BrowseName	CertificateDirectoryType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *DirectoryType* defined in 6.3.3.					
Organizes	Object	CertificateGroups		CertificateGroup FolderType	Mandatory
HasComponent	Method	StartSigningRequest	Defined in 7.6.3		Mandatory
HasComponent	Method	StartNewKeyPairRequest	Defined in 7.6.4		Mandatory
HasComponent	Method	FinishRequest	Defined in 7.6.5		Mandatory
HasComponent	Method	GetCertificateGroups	Defined in 7.6.6		Mandatory
HasComponent	Method	GetTrustList	Defined in 7.6.6		Mandatory
HasComponent	Method	GetCertificateStatus	Defined in 7.6.8		Mandatory

The *CertificateGroups Object* organizes the *Certificate Groups* supported by the *CertificateManager*. It is described in 7.5.17. *CertificateManagers* shall support the *DefaultApplicationGroup* and may support the *DefaultHttpsGroup* or the *DefaultUserTokenGroup*. *CertificateManagers* may support additional *Certificate Groups* depending on their requirements. For example, a *CertificateManager* with multiple Certificate Authorities would represent each as a *CertificateGroupType* Object organized by *CertificateGroups Folder. Clients* could then request *Certificates* issued by a specific CA by passing the appropriate *NodeId* to the *StartSigningRequest* or *StartNewKeyPairRequest Methods*.

The *StartSigningRequest Method* is used to request a new a *Certificate* that is signed by a CA managed by the *CertificateManager*. This *Method* is recommended when the caller already has a private key.

The *StartNewKeyPairRequest Method* is used to request a new *Certificate* that is signed by a CA managed by the *CertificateManager* along with a new private key. This *Method* is used only when the caller does not have a private key and cannot generate one.

The *FinishRequest Method* is used to check that a *Certificate* request has been approved by the *CertificateManager Administrator*. If successful the *Certificate* and *Private Key* (if requested) are returned.

The *GetCertificateGroups Method* returns a list of *NodeIds* for *CertificateGroupType Objects* that can be used to request *Certificates* or *Trust Lists* for an *Application*.

The *GetTrustList Method* returns a *NodeId* of a *TrustListType Object* that can be used to read a *Trust List* for an *Application*.

The *GetCertificateStatus Method* checks whether the *Application* needs to update its *Certificate*.

7.6.3 StartSigningRequest

StartSigningRequest is used to initiate a request to create a *Certificate* that uses the private key that the caller currently has. The new *Certificate* is returned in the *FinishRequest* response.

Signature

```
StartSigningRequest(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [in]  NodeId certificateTypeId
      [in]  ByteString certificateRequest
      [out] NodeId requestId
     );
```

Argument	Description
applicationId	The identifier assigned to the *Application* record by the *CertificateManager*.
certificateGroupId	The *NodeId* of the Certificate Group which provides the context for the new request. If null the *CertificateManager* shall choose the *DefaultApplicationGroup*.
certificateTypeId	The *NodeId* of the *CertificateType* for the new *Certificate*. If null the *CertificateManager* shall generate a *Certificate* based on the value of the certificateGroupId argument.
certificateRequest	A *CertificateRequest is* used to prove possession of the *Private Key*. It is a PKCS #10 encoded blob in DER format. If the *CertificateRequest* is for an *ApplicationInstance Certificate* then it shall include all fields required by IEC 62541-6 such as the *subjectAltName*.
requestId	The *NodeId* that represents the request. This value is passed to *FinishRequest*.

The call returns the *NodeId* that is passed to the *FinishRequest Method*.

The *certificateGroupId* parameter allows the caller to specify a *Certificate Group* that provides context for the request. If null the *CertificateManager* shall choose the *DefaultApplicationGroup*. The set of available *Certificate Groups* are found in the *CertificateGroups* folder described in 7.6.2. The *Certificate Groups* allowed for an *Application* are returned by the *GetCertificateGroups Method* (see 7.6.6).

The *certificateTypeId* parameter specifies the type of *Certificate* to return. The permitted values are specified by the CertificateTypes Property of the Object specified by the certificateGroupId parameter.

The *certificateRequest* parameter is a DER encoded *Certificate Request*. The subject name, subject alternative name and public key are copied into the new *Certificate*.

If the *certificateTypeId* is a subtype of *ApplicationCertificateType* the subject name shall have an organization (O=) or domain name (DC=) field. The public key length shall meet the length restrictions for the *CertificateType.* If the *certificateType* is a subtype of *HttpsCertificateType* the *Certificate* common name (CN=) shall be the same as a domain from a *DiscoveryUrl* which uses HTTPS and the subject name shall have an organization (O=) field. The public key length shall be greater than or equal to 1 024 bits.

The *ApplicationUri* shall be specified in the CSR. The *CertificateManager* shall return *Bad_CertificateUriInvalid* if the stored *ApplicationUri* for the Application is different from what is in the CSR.

For *Servers*, the list of domain names shall be specified in the CSR. The domains shall include the domain(s) in the *DiscoveryUrls* known to the *CertificateManager*.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions. It can also be invoked by the *Application* that owns the private key used to sign the *CertificateRequest* (e.g. the private key shall be the private key used to create the *SecureChannel*).

If auditing is supported, the *CertificateManager* shall generate the *CertificateRequestedAuditEventType* (see 7.6.9) if this *Method* succeeds or fails.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationId* does not refer to a registered *Application*.
Bad_InvalidArgument	The certificateGroupId, *certificateTypeId* or *certificateRequest* is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_RequestNotAllowed	The current configuration of the *CertificateManager* does not allow the request. The text associated with the error should indicate the exact reason.
Bad_CertificateUriInvalid	The ApplicationUri was not specified in the CSR or does not match the Application record.
Bad_NotSupported	The signing algorithm, public algorithm or public key size are not supported by the *CertificateManager*. The text associated with the error shall indicate the exact problem.

Table 31 specifies the *AddressSpace* representation for the *StartSigningRequest Method*.

Table 31 – StartSigningRequest Method AddressSpace definition

Attribute	Value				
BrowseName	StartSigningRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.4 StartNewKeyPairRequest

This *Method* is used to start a request for a new *Certificate* and *Private Key*. The *Certificate* and private key are returned in the *FinishRequest* response.

Signature

```
StartNewKeyPairRequest(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [in]  NodeId certificateTypeId
      [in]  String subjectName
      [in]  String[] domainNames
      [in]  String privateKeyFormat
      [in]  String privateKeyPassword
       [out] NodeId requestId
    );
```

Argument	Description
applicationId	The identifier assigned to the *Application Instance* by the *CertificateManager*.
certificateGroupId	The *NodeId* of the Certificate Group which provides the context for the new request. If null the *CertificateManager* shall choose the *DefaultApplicationGroup*.
certificateTypeId	The *NodeId* of the *CertificateType* for the new *Certificate*. If null the *CertificateManager* shall generate a *Certificate* based on the value of the certificateGroupId argument.
subjectName	The subject name to use for the *Certificate*. If not specified the *ApplicationName* and/or *domainNames* are used to create a suitable default value. The format of the subject name is a sequence of name value pairs separated by a "/". The name shall be one of "CN", "O", "OU", "DC", "L", "S" or "C" and shall be followed by a "=" and then followed by the value. The value may be any printable character except for ". If the value contains a "/" or a "=", then it shall be enclosed in double quotes (").
domainNames	The domain names to include in the *Certificate*. If not specified the *DiscoveryUrls* are used to create suitable defaults.
privateKeyFormat	The format of the private key. The following values are always supported: PFX - PKCS #12 encoded PEM - Base64 encoded DER (see IETF RFC 5958).
privateKeyPassword	The password to use for the private key.
requestId	The *NodeId* that represents the request. This value is passed to *FinishRequest*.

The call returns the *NodeId* that is passed to the *FinishRequest Method*.

The *certificateGroupId* parameter allows the caller to specify a *Certificate Group* that provides context for the request. If null, the *CertificateManager* shall choose the *DefaultApplicationGroup*. The set of available *Certificate Groups* are found in the *CertificateGroups* folder described in 7.6.2. The *Certificate Groups* allowed for an *Application* are returned by the *GetCertificateGroups Method* (see 7.6.6).

The *certificateTypeId* parameter specifies the type of *Certificate* to return. The permitted values are specified by the *CertificateTypes Property* of the *Object* specified by the certificateGroupId parameter.

The *subjectName* parameter is a sequence of X.500 name value pairs separated by a "/". For example: CN=ApplicationName/OU=Group/O=Company.

If the *certificateType* is a subtype of *ApplicationCertificateType* the *Certificate* subject name shall have an organization (O=) or domain name (DC=) field. The public key length shall meet the length restrictions for the *CertificateType.* The domain name field specified in the subject name is a logical domain used to qualify the subject name that may or may not be the same as a domain or IP address in the subjectAltName field of the *Certificate*.

If the *certificateType* is a subtype of *HttpsCertificateType* the *Certificate* common name (CN=) shall be the same as a domain from a *DiscoveryUrl* which uses HTTPS and the subject name shall have an organization (O=) field.

If the subjectName is blank or null, the *CertificateManager* generates a suitable default.

The *domainNames* parameter is list of domains to be includes in the *Certificate*. If it is null or empty, the GDS uses the *DiscoveryUrls* of the *Server* to create a list. For *Clients*, the *domainNames* are omitted from the *Certificate* if they are not explicitly provided.

The *privateKeyFormat* specifies the format of the private key returned. All *CertificateManager* implementations shall support "PEM" and "PFX".

The *privateKeyPassword* specifies the password on the private key. The *CertificateManager* shall not persist this information and shall discard it once the new private key is generated.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions.

If auditing is supported, the *CertificateManager* shall generate the *CertificateRequested AuditEventType* (see 7.6.9) if this *Method* succeeds or fails.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NodeIdUnknown	The *applicationId* does not refer to a registered *Application*.
Bad_InvalidArgument	The certificateGroupId, *certificateTypeId*, *subjectName*, *domainNames* or *privateKeyFormat* parameter is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_RequestNotAllowed	The current configuration of the CertificateManager does not allow the request. The text associated with the error should indicate the exact reason.

Table 32 specifies the *AddressSpace* representation for the *StartNewKeyPairRequest Method*.

Table 32 – StartNewKeyPairRequest Method AddressSpace definition

Attribute	Value				
BrowseName	StartNewKeyPairRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.5 FinishRequest

FinishRequest is used to finish a certificate request started with a call to *StartNewKeyPairRequest* or *StartSigningRequest*.

Signature

```
FinishRequest (
      [in]  NodeId applicationId
      [in]  NodeId requestId
      [out] ByteString certificate
      [out] ByteString privateKey
      [out] ByteString[] issuerCertificates
    );
```

Argument	Description
applicationId	The identifier assigned to the *Application Instance* by the GDS.
requestId	The *NodeId* returned by *StartNewKeyPairRequest* or *StartSigningRequest*.
certificate	The DER encoded *Certificate*.
privateKey	The private key encoded in the format requested. If a password was supplied, the blob is protected with it. This field is null if no private key was requested.
issuerCertificates	The *Certificates* required to validate the new *Certificate*.

This call is passes the *NodeId* returned by a previous call to *StartNewKeyPairRequest* or *StartSigningRequest*.

It is expected that a *Client* will periodically call this *Method* until the GDS has approved the request.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions. It can also be invoked by the *Application* that owns the *Certificate* (e.g. the private key used to create the channel shall be the same as the private key used to sign the request passed to *StartSigningRequest*).

The *Method* shall only be called via a *SecureChannel* with encryption enabled.

If auditing is supported, the GDS shall generate the *CertificateDeliveredAuditEventType* (see 7.6.10) if this *Method* succeeds or if it fails with anything but *Bad_NothingToDo*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationId* does not refer to a registered *Application*.
Bad_InvalidArgument	The *requestId* is does not reference to a valid request for the *Application*.
Bad_NothingToDo	There is nothing to do because request has not yet completed.
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_RequestNotAllowed	The *CertificateManager* rejected the request. The text associated with the error should indicate the exact reason.

Table 33 specifies the *AddressSpace* representation for the *FinishRequest Method*.

Table 33 – FinishRequest Method AddressSpace definition

Attribute	Value				
BrowseName	FinishRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.6 GetCertificateGroups

GetCertificateGroups returns the Certificate Groups assigned to *Application*.

Signature

```
GetCertificateGroups(
      [in]  NodeId   applicationId
      [out] NodeId[] certificateGroupIds
      );
```

Argument	Description
applicationId	The identifier assigned to the *Application* by the GDS.
certificateGroupIds	An identifier for the Certificate Groups assigned to the Application.

A *Certificate Group* provides a *Trust List* and one or more *CertificateTypes* which may be assigned to an *Application*. The values returned by this Method are passed to the *GetTrustList* (see 7.6.7), *StartSigningRequest* (see 7.6.3) or *StartNewKeyPairRequest* (see 7.6.4) *Methods*.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions. It can also be invoked by the *Application* identified by the *applicationId* (e.g. the private key used to create the channel shall be private key associated with the *Certificate* assigned to the *Application*).

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationId* does not refer to a registered *Application*.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 34 specifies the *AddressSpace* representation for the *GetCertificateGroups Method*.

Table 34 – GetCertificateGroups Method AddressSpace definition

Attribute	Value				
BrowseName	GetCertificateGroups				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.7 GetTrustList

GetTrustList is used to retrieve the *NodeId* of a *Trust List* assigned to an *Application*.

Signature

```
GetTrustList(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [out] NodeId trustListId
      );
```

Argument	Description
applicationId	The identifier assigned to the *Application* by the GDS.
certificateGroupId	An identifier for a *Certificate Group* that the *Application* belongs to. If null the *CertificateManager* shall return the *trustListId* for a suitable default group for the *Application*.
trustListId	The *NodeId* for a *Trust List Object* that can be used to download the *Trust List* assigned to the *Application*.

Access permissions also apply to the *Trust List Objects* which are returned by this *Method*. This *Trust List* includes any *Certificate Revocation Lists* (CRLs) associated with issuer *Certificates* in the *Trust List*.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions. It can also be invoked by the *Application* identified by the *applicationId* (e.g. the private key used to create the channel shall be the private key associated with the *Certificate* assigned to the *Application*).

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationId* does not refer to a registered *Application*.
Bad_InvalidArgument	The certificateGroupId parameter is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 35 specifies the *AddressSpace* representation for the *GetTrustList Method*.

Table 35 – GetTrustList Method AddressSpace definition

Attribute	Value				
BrowseName	GetTrustList				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.8 GetCertificateStatus

GetCertificateStatus is used to check if an *Application* needs to update its *Certificate*.

Signature

```
GetCertificateStatus(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId

      [in]  NodeId certificateTypeId
      [out] Boolean updateRequired
      );
```

Argument	Description
applicationId	The identifier assigned to the *Application Instance* by the GDS.
certificateGroupId	The *NodeId* of the Certificate Group that provides the context. If null the *CertificateManager* shall choose the *DefaultApplicationGroup*.
certificateTypeId	The *NodeId* of the *CertificateType* for the *Certificate*. If null the *CertificateManager* shall select a *Certificate* based on the value of the certificateGroupId argument.
updateRequired	TRUE if the *Application* needs to request a new *Certificate* from the GDS. FALSE if the *Application* can keep using the existing *Certificate*.

Access permissions that apply to *CreateSigningRequest Method* shall apply to this *Method*.

This *Method* can be invoked by a configuration tool which has provided user credentials with necessary access permissions. It can also be invoked by the *Application* identified by the *applicationId* (e.g. the private key used to create the channel shall be private key associated with the *Certificate* assigned to the *Application*).

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationId* does not refer to a registered *Application*.
Bad_InvalidArgument	The *certificateGroupId* or *certificateTypeId* parameter is not valid. The text associated with the error shall indicate the exact problem.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 36 specifies the *AddressSpace* representation for the *GetCertificateStatus Method*.

Table 36 – GetCertificateStatus Method AddressSpace definition

Attribute	Value				
BrowseName	GetCertificateStatus				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.9 CertificateRequestedAuditEventType

This event is raised when a new certificate request has been accepted or rejected by the GDS.

This can be the result of a *StartNewKeyPairRequest* or *StartSigningRequest Method* calls.

Its representation in the *AddressSpace* is formally defined in Table 37.

Table 37 – CertificateRequestedAuditEventType definition

Attribute	Value				
BrowseName	CertificateRequestedAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Mandatory
HasProperty	Variable	CertificateType	NodeId	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

The *CertificateGroup Property* specifies the *Certificate Group* that was affected by the update.

The *CertificateType Property* specifies the type of *Certificate* that was updated.

7.6.10 CertificateDeliveredAuditEventType

This event is raised when a certificate is delivered by the GDS to a *Client*.

This is the result of a *FinishRequest Method* completing successfully.

Its representation in the *AddressSpace* is formally defined in Table 38.

Table 38 – CertificateDeliveredAuditEventType definition

Attribute	Value				
BrowseName	CertificateDeliveredAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Mandatory
HasProperty	Variable	CertificateType	NodeId	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

The *CertificateGroup Property* specifies the *Certificate Group* that was affected by the update.

The *CertificateType Property* specifies the type of *Certificate* that was updated.

7.7 Information Model for Push Certificate Management

7.7.1 Overview

If a *Server* supports Push Management, it is required to support an information model as part of its address space. It shall support the *ServerConfiguration Object* shown in Figure 15. This *Object* shall only be visible and accessible to administrators and/or the GDS.

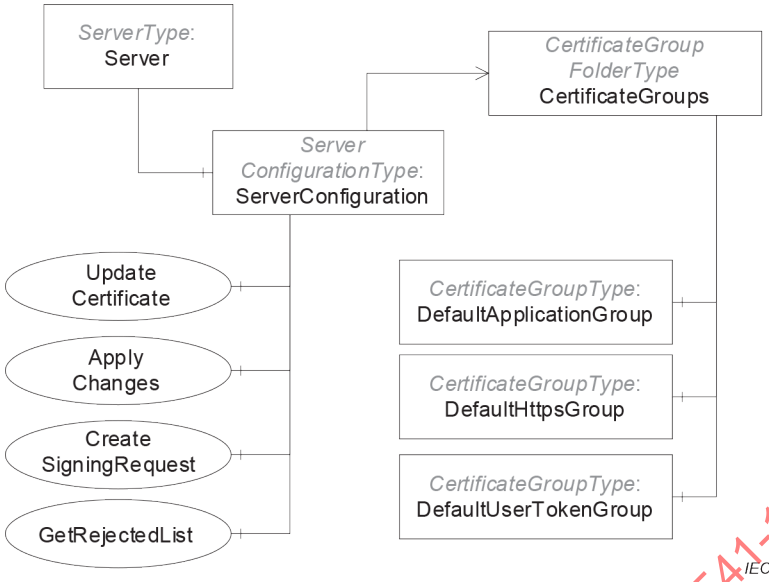


Figure 15 – The AddressSpace for the Server that supports Push Management

All access to *Methods* defined on the *ServerConfiguration Object* shall be over an encrypted channel. In addition, Servers should have user credentials with administrator privileges.

7.7.2 ServerConfiguration

This *Object* allows access to the *Server's* configuration and it is the target of an *HasComponent* reference from the *Server Object* defined in IEC 62541-5.

This *Object* and its immediate children shall be visible (i.e. browse access is available) to users who can access the *Server Object*. The children of the *CertificateGroups Object* should only be visible to authorized administrators.

Its representation in the *AddressSpace* is formally defined in Table 39.

Table 39 – ServerConfiguration Object definition

Attribute	Value				
BrowseName	ServerConfiguration				
Namespace	CORE (see 3.3)				
TypeDefinition	ServerConfigurationType defined in 7.7.3.				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**

7.7.3 ServerConfigurationType

This type defines an *ObjectType* that represents the configuration of a *Server* that supports Push Management. Its values are defined in Table 40. There is always exactly one instance in the *Server AddressSpace*.

Table 40 – ServerConfigurationType definition

Attribute	Value				
BrowseName	ServerConfigurationType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**Type Definition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasComponent	Object	CertificateGroups		CertificateGroup FolderType	Mandatory
HasProperty	Variable	ServerCapabilities	String[]	PropertyType	Mandatory
HasProperty	Variable	SupportedPrivateKeyFormats	String[]	PropertyType	Mandatory
HasProperty	Variable	MaxTrustListSize	UInt32	PropertyType	Mandatory
HasProperty	Variable	MulticastDnsEnabled	Boolean	PropertyType	Mandatory
HasComponent	Method	UpdateCertificate	See 7.7.4		Mandatory
HasComponent	Method	ApplyChanges	See 7.7.5		Optional
HasComponent	Method	CreateSigningRequest	See 7.7.6		Mandatory
HasComponent	Method	GetRejectedList	See 7.7.7		Mandatory

The *CertificateGroups Object* organizes the *Certificate Groups* supported by the *Server*. It is described in 7.5.17. *Servers* shall support the *DefaultApplicationGroup* and may support the *DefaultHttpsGroup* or the *DefaultUserTokenGroup*. *Servers* may support additional *Certificate Groups* depending on their requirements. For example, a *Server* with two network interfaces should have a different *Trust List* for each interface. The second *Trust List* would be represented as a new *CertificateGroupType* Object organized by *CertificateGroups Folder.*

The *ServerCapabilities Property* specifies the capabilities from Annex D which the *Server* supports. The value is the same as the value reported to the *LocalDiscoveryServer* when the *Server* calls the *RegisterServer2 Service*.

The *SupportedPrivateKeyFormats* specifies the *PrivateKey* formats supported by the Server. Possible values include "PEM" (see IETF RFC 5958) or "PFX" (see PKCS #12). The array is empty if the *Server* does not allow external *Clients* to update the *PrivateKey*.

The *MaxTrustListSize* is the maximum size of the *Trust List* in bytes. 0 means no limit. The default is 65 535 bytes.

If *MulticastDnsEnabled* is TRUE then the *Server* announces itself using multicast DNS. It can be changed by writing to the *Variable*.

The *GetRejectedList Method* returns the list of *Certificates* that have been rejected by the *Server*. It can be used to track activity or allow administrators to move a rejected *Certificate* into the *Trust List*.

The *UpdateCertificate Method* is used to update a *Certificate*.

The *ApplyChanges Method* is used to apply any security related changes if the *Server* sets the *applyChangesRequired* flag when another *Method* is called. *Servers* should minimize the impact of applying the new configuration; however, it could require that all existing *Sessions* be closed and re-opened by the *Clients*.

The *CreateSigningRequest Method* asks the *Server* to create a *PKCS #10* encoded *Certificate Request* that is signed with the *Server's* private key.

7.7.4 UpdateCertificate

UpdateCertificate is used to update a Certificate for a Server.

There are the following three use cases for this *Method*:

- the new *Certificate* was created based on a signing request created with the *Method CreateSigningRequest* defined in 7.7.6. In this case, there is no *privateKey* provided;
- a new *privateKey* and *Certificate* was created outside the *Server* and both are updated with this *Method*;
- a new *Certificate* was created and signed with the information from the old *Certificate*. In this case there is no *privateKey* provided.

The *Server* shall do all normal integrity checks on the *Certificate* and all of the issuer *Certificates*. If errors occur the *Bad_SecurityChecksFailed* error is returned.

The *Server* shall report an error if the public key does not match the existing *Certificate* and the privateKey was not provided.

If the *Server* returns *applyChangesRequired*=FALSE then it is indicating that it is able to satisfy the requirements specified for the *ApplyChanges Method*.

This *Method* requires an encrypted channel and that the Client provides credentials with administrative rights on the Server.

Signature

```
UpdateCertificate(
          [in] NodeId certificateGroupId
          [in] NodeId certificateTypeId
          [in] ByteString certificate
          [in] ByteString[] issuerCertificates
          [in] String privateKeyFormat
          [in] ByteString privateKey
            [out] Boolean applyChangesRequired
    );
```

Argument	Description
certificateGroupId	The NodeId of the *Certificate Group Object* that is affected by the update. If null the *DefaultApplicationGroup* is used.
certificateTypeId	The type of *Certificate* being updated. The set of permitted types is specified by the *CertificateTypes Property* belonging to the *Certificate Group*.
certificate	The DER encoded *Certificate* which replaces the existing Certificate.
issuerCertificates	The issuer *Certificates* needed to verify the signature on the new *Certificate*.
privateKeyFormat	The format of the *Private Key* (PEM or PFX). If the *privateKey* is not specified the *privateKeyFormat* is null or empty.
privateKey	The *Private Key* encoded in the *privateKeyFormat*.
applyChangesRequired	Indicates that the *ApplyChanges Method* shall be called before the new *Certificate* will be used.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The certificateTypeId or certificateGroupId is not valid.
Bad_CertificateInvalid	The *Certificate* is invalid or the format is not supported.
Bad_NotSupported	The *PrivateKey* is invalid or the format is not supported.
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_SecurityChecksFailed	Some failure occurred verifying the integrity of the *Certificate*.

Table 41 specifies the *AddressSpace* representation for the *UpdateCertificate Method*.

Table 41 – UpdateCertificate Method AddressSpace Definition

Attribute	Value				
BrowseName	UpdateCertificate				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.7.5 ApplyChanges

ApplyChanges is used to tell the *Server* to apply any security changes.

This *Method* should only be called if a previous call to a *Method* that changed the configuration returns applyChangesRequired=true (see 7.7.4).

If the *Server Certificate* has changed, *Secure Channels* using the old *Certificate* will eventually be interrupted. The only leeway the *Server* has is with the timing. In the best case, the *Server* can close the *TransportConnections* for the affected *Endpoints* and leave any *Subscriptions* intact. This should appear no different than a network interruption from the perspective of the *Client*. The *Client* should be prepared to deal with *Certificate* changes during its reconnect logic. In the worst case, a full shutdown which affects all connected *Clients* will be necessary. In the latter case, the *Server* shall advertise its intent to interrupt connections by setting the *SecondsTillShutdown* and *ShutdownReason Properties* in the *ServerStatus Variable*.

If the *Secure Channel* being used to call this *Method* will be affected by the *Certificate* change then the *Server* shall introduce a delay long enough to allow the caller to receive a reply.

This *Method* requires an encrypted channel and that the *Client* provide credentials with administrative rights on the *Server*.

Signature

```
ApplyChanges();
```

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 42 specifies the *AddressSpace* representation for the *ApplyChanges Method*.

Table 42 – ApplyChanges Method AddressSpace Definition

Attribute	Value				
BrowseName	ApplyChanges				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.7.6 CreateSigningRequest

CreateSigningRequest Method asks the *Server* to create a *PKCS #10* DER encoded *Certificate Request* that is signed with the *Server's* private key. This request can be then used to request a *Certificate* from a CA that expects requests in this format.

This *Method* requires an encrypted channel and that the *Client* provide credentials with administrative rights on the *Server*.

Signature

```
CreateSigningRequest(
    [in] NodeId certificateGroupId,
    [in] NodeId certificateTypeId,
    [in] String subjectName,
    [in] Boolean regeneratePrivateKey,
    [in] ByteString nonce,
    [out]    ByteString certificateRequest
    );
```

Argument	Description
certificateGroupId	The NodeId of the *Certificate Group Object* which is affected by the request. If null the *DefaultApplicationGroup* is used.
certificateTypeId	The type of *Certificate* being requested. The set of permitted types is specified by the *CertificateTypes Property* belonging to the *Certificate Group*.
subjectName	The subject name to use in the *Certificate Request*. If not specified the *SubjectName* from the current *Certificate* is used. The format of the *subjectName* is defined in 7.6.4.
regeneratePrivateKey	If TRUE the *Server* shall create a new *Private Key* which it stores until the matching signed *Certificate* is uploaded with the *UpdateCertificate Method*. Previously created *Private Keys* may be discarded if *UpdateCertificate* was not called before calling this method again. If FALSE the *Server* uses its existing *Private Key*.
nonce	Additional entropy which the caller shall provide if regeneratePrivateKey is TRUE. It shall be at least 32 bytes long.
certificateRequest	The PKCS #10 DER encoded *Certificate Request*. If the *CertificateRequest* is for an *ApplicationInstance Certificate* then it shall include all fields required by IEC 62541-6, such as the *subjectAltName*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The *certificateTypeId, certificateGroupId* or *subjectName* is not valid.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 43 specifies the *AddressSpace* representation for the *CreateSigningRequest Method*.

Table 43 – CreateSigningRequest Method AddressSpace definition

Attribute	Value				
BrowseName	CreateSigningRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.7.7 GetRejectedList

GetRejectedList Method returns the list of *Certificates* that have been rejected by the *Server*.

No rules are defined for how the *Server* updates this list or how long a *Certificate* is kept in the list. It is recommended that every valid but untrusted *Certificate* be added to the rejected list as long as storage is available. *Servers* should omit older entries from the list returned if the maximum message size is not large enough to allow the entire list to be returned.

This *Method* requires an encrypted channel and that the *Client* provide credentials with administrative rights on the *Server*.

Signature

```
GetRejectedList(
    [out] ByteString[] certificates
    );
```

Argument	Description
certificates	The DER encoded form of the Certificates rejected by the Server.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 44 specifies the *AddressSpace* representation for the *GetRejectedList Method*.

Table 44 – GetRejectedList Method AddressSpace definition

Attribute	Value				
BrowseName	GetRejectedList				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.7.8 CertificateUpdatedAuditEventType

This event is raised when the *Application Certificate* is changed.

This is the result of a *UpdateCertificate Method* completing successfully or failing.

Its representation in the *AddressSpace* is formally defined in Table 45.

Table 45 – CertificateUpdatedAuditEventType definition

Attribute	Value				
BrowseName	CertificateUpdatedAuditEventType				
Namespace	CORE (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Mandatory
HasProperty	Variable	CertificateType	NodeId	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

The *CertificateGroup Property* specifies the *Certificate Group* that was affected by the update.

The *CertificateType Property* specifies the type of *Certificate* that was updated.

8 KeyCredential management

8.1 Overview

KeyCredential management functions allow the management and distribution of *KeyCredentials* which *OPC UA Applications* use to access *Authorization Services* and/or *Brokers*. An application that provides the *KeyCredential* management functions is called a *KeyCredentialService* and is typically combined with the GDS into a single application.

There are two primary models for *KeyCredential* management: pull and push management. In pull management, the application acts as a *Client* and uses the *Methods* on the *KeyCredentialService* to request and update *KeyCredentials*. The application is responsible for ensuring the *KeyCredentials* are kept up to date. In push management the application acts as a *Server* and exposes *Methods* that the *KeyCredentialService* can call to update the *KeyCredentials* as required.

A *KeyCredentialService* can directly manage the *KeyCredentials* it supplies or it may act as an intermediary between a *Client* and a system that does not support OPC UA, such as Azure AD[1] or LDAP.

Note that *KeyCredentials* are secrets that are directly passed to *Authorization Services* and/or *Brokers* and are not *Certificates* with private keys. *Certificate* distribution is managed by the *Certificate* management model described in 7. For example, *Authorization Service*s that support OAuth2 often require the client to provide a client_id and client_secret parameter with any request. The *KeyCredentials* are the values that the application shall place in these parameters.

[1] Azure AD is the name of a product supplied by Microsoft®. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

8.2 Pull management

Pull management is performed by using a *KeyCredentialManagement Object* (see 8.4.3). It allows *Clients* to request credentials for *Authorization Services* or *Brokers* which are supported by the *KeyCredentialService*. The interactions between the *Client* and the *KeyCredentialService* during pull management are illustrated in Figure 16.

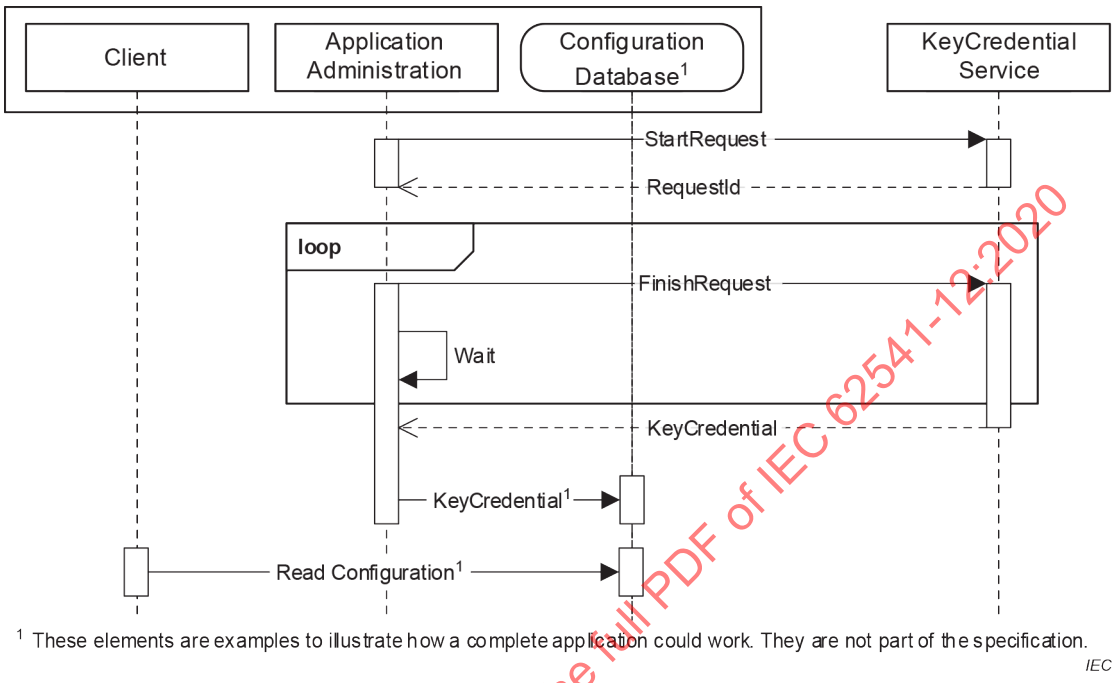


Figure 16 – The Pull Model for KeyCredential management

The Application Administration component may be part of the *Client* or a standalone utility that understands how the *Client* persists its configuration information in its Configuration Database. The administration and database components are examples to illustrate how an application could be built and are not a requirement.

Requesting credentials is a two stage process because some *KeyCredentialServices* require a human to review and approve requests. The calls to the *FinishKeyCredentialRequest Method* may not be periodic and could be initiated by events such as a user starting up the application or interacting with a UI element such as a button.

KeyCredentials can only be requested for *Clients* which are trusted by the *KeyCredentialService*.

Security in pull management requires an encrypted channel and the use of administrator credentials for the *KeyCredentialService* that ensure only authorized users can request *KeyCredentials*.

8.3 Push management

Push management is performed by using a *KeyCredentialConfiguration Object* (see 8.5.3) which is a component of the *KeyCredentialManagement Folder* which is a component of the *ServerConfiguration Object* in a *Server*. The interactions between the Administration application and the *KeyCredentialService* during push management are illustrated in Figure 17.

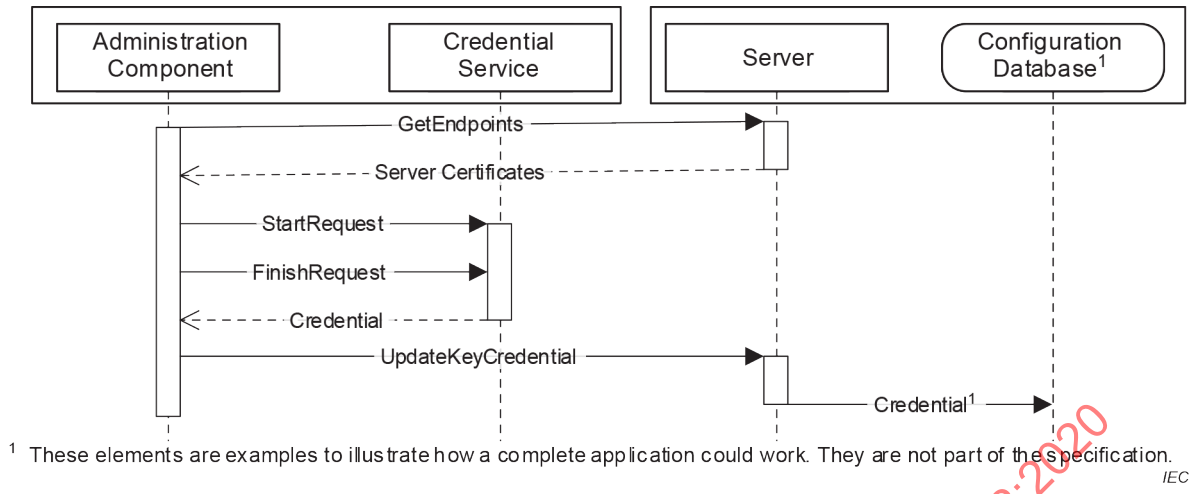


Figure 17 – The Push Model for KeyCredential management

The Administration Component may use internal APIs to manage *KeyCredentials* or it could be a standalone utility that uses OPC UA to communicate with a *Server* which supports the pull model (see 8.2). The Configuration Database is used by the *Server* to persist its configuration information. The administration and database components are examples to illustrate how an application could be built and are not a requirement.

To ensure security of the *KeyCredentials,* the *KeyCredentialService* component can require that secrets be encrypted with a key only known to the intended recipient of the *KeyCredentials*. For this reason, the Administration Component uses the *GetEndpoints Service* to read the *Certificate* from the *Server* before initiating the credential request on behalf of the *Server*.

Security, when using the push management model, requires an encrypted channel and the use of administrator credentials for the *Server* that ensure only authorized users can update *KeyCredentials*. If the KeyCredentialService component is separate from the administration component, then different administrator credentials are required for the *Server* that ensure that only authorized users can request new *KeyCredentials* on behalf of *Servers*.

8.4 Information Model for pull management

8.4.1 Overview

The *AddressSpace* used for pull management is shown in Figure 18. *Clients* interact with the *Nodes* defined in this model when they need to request or revoke *KeyCredentials* for themselves or for another application. The *KeyCredentialManagement Folder* is a well-known *Object* that appears in the *AddressSpace* of any *Server* which supports *KeyCredential* management.

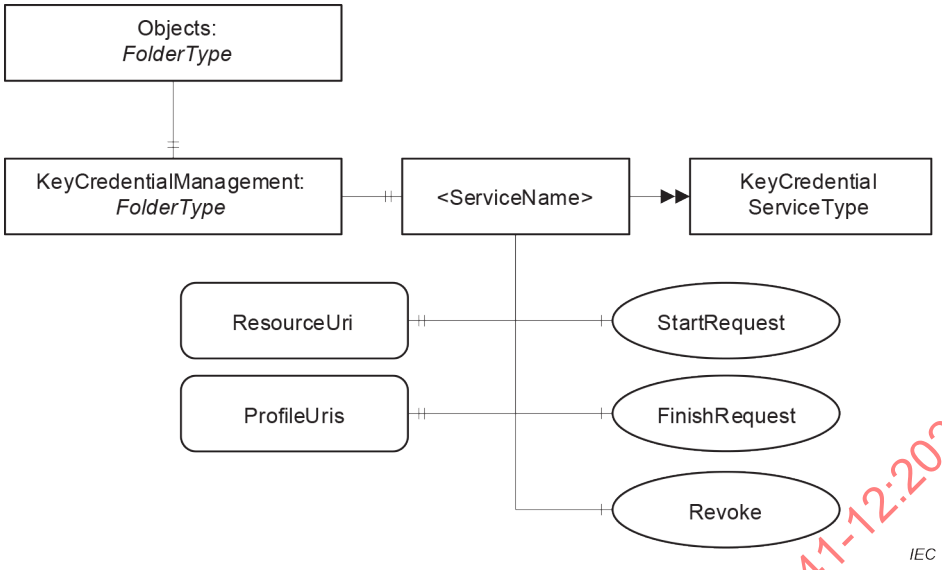


Figure 18 – The Address Space used for Pull KeyCredential management

8.4.2 KeyCredentialManagement

This *Object* is an instance of *FolderType*. It contains the *KeyCredentialService Objects* which may be accessed via the *Server*. It is the target of an *Organizes* reference from the *Objects Folder* defined in IEC 62541-5. It is defined in Table 46.

Table 46 – KeyCredentialManagement Object definition

Attribute	Value			
BrowseName	KeyCredentialManagement			
Namespace	GDS (see 3.3)			
TypeDefinition	FolderType defined in IEC 62541-5.			
References	**NodeClass**	**BrowseName**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<ServiceName>	KeyCredentialServiceType	OptionalPlaceholder

8.4.3 KeyCredentialServiceType

This *ObjectType* is the *TypeDefinition* for an *Object* that allows the management of *KeyCredentials*. It is defined in Table 47.

Table 47 – KeyCredentialServiceType definition

Attribute	Value				
BrowseName	KeyCredentialServiceType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	ProfileUris	String[]	PropertyType	Mandatory
HasComponent	Method	StartRequest		Defined in 8.4.4.	Mandatory
HasComponent	Method	FinishRequest		Defined in 8.4.5.	Mandatory
HasComponent	Method	Revoke		Defined in 8.4.6.	Optional

The *ResourceUri Property* uniquely identifies the resource that accepts the *KeyCredentials* provided by the *KeyCredentialService Object*.

The *ProfileUris Property* specifies URIs assigned in IEC 62541-7 to the authentication mechanism used to communicate with the resource that accepts *KeyCredentials* provided by the *Object*. For example, it could specify that the resource returns JWTs using OAuth2 HTTP based APIs. As another example, it could specify an MQTT broker that expects a username/password.

The *StartRequest Method* is used to initiate a request for new *KeyCredentials* for an application. This request can complete immediately, or it can require offline approval by an administrator.

The *FinishRequest Method* is used to complete a request created by calling *StartRequest*. If the *KeyCredential* is available, it is returned. If request is not yet completed it returns *Bad_NothingToDo*.

The *Revoke Method* is used to revoke a previously issued *KeyCredential*.

8.4.4 StartRequest

StartRequest is used to request a new *KeyCredential*.

The *KeyCredential* secret may be encrypted with the public key of the *Certificate* supplied in the request. The *SecurityPolicyUri* specifies the security profile used for the encryption.

This *Method* requires an encrypted channel and that the *Client* provides credentials with administrative rights for the application requesting the credentials.

Signature

```
StartRequest (
      [in]  String         applicationUri,
      [in]  ByteString publicKey,
      [in]  String         securityPolicyUri,
      [in]  NodeId[]    requestedRoles,
      [out] NodeId         requestId
    );
```

Argument	Description
applicationUri	The *applicationUri* of the application receiving the *KeyCredentials*. The request is rejected *applicationUri* does not uniquely identify an application known to the GDS (see 6.3.6). If the requestor is not the same as the application used to create the *Secure Channel*, then a *Certificate* should be provided.
publicKey	A *Public Key* used to encrypt the returned *KeyCredential* secret. For RSA *SecurityPolicies*, this is the DER encoded form of an X.509 v3 *Certificate* as described in IEC 62541-6. For ECC *SecurityPolicies*, this is an ephemeral key created by the owner of the *KeyCredentials*. Not specified if no encryption is required. If the *securityPolicyUri* is provided, this field shall be provided.
securityPolicyUri	The *SecurityPolicy* used to encrypt the secret. If the *certificate* is provided, this field shall be provided.
requestedRoles	A list of *Roles* which should be assigned to the *KeyCredential*. If not provided, the *Server* chooses suitable defaults. The *Server* ignores *Roles* which it does not recognize or if the caller is not authorized to request access to the *Role*.
requestId	A unique identifier for the request. This identifier shall be passed to the *FinishRequest* (see 8.4.5).

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_NotFound	The *applicationUri* is not known to the GDS.
Bad_ConfigurationError	The *applicationUri* is used by multiple records in the GDS.
Bad_CertificateInvalid	The *Certificate* is invalid.
Bad_SecurityPolicyRejected	The *SecurityPolicy* is unrecognized or not allowed or does not match the *Certificate*.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 48 specifies the *AddressSpace* representation for the *StartRequest Method*.

Table 48 – StartRequest Method AddressSpace definition

Attribute	Value				
BrowseName	StartRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

8.4.5 FinishRequest

FinishRequest is used to retrieve a *KeyCredential*.

If a *Certificate* was provided in the request then the *KeyCredential* secret is encrypted using an asymmetric encryption algorithm specified by the *SecurityPolicyUri* provided in the request.

The format of the signed and encrypted *credentialSecret* is the same as the Version 2 Token Secret Format defined in IEC 62541-4. When used for the *credentialSecret*, the signature is provided by the source of the *KeyCredential*, which can be the GDS *Application Instance Certificate*.

If the return code is *Bad_RequestNotComplete*, then the request has not been processed and the *Client* should call again. The recommended time between calls depends on the GDS.

This *Method* requires an encrypted channel and that the *Client* provides credentials with administrative rights for the application requesting the credentials.

Signature

```
FinishRequest (
      [in]  NodeId     requestId,
      [in]  Boolean    cancelRequest,
      [out] String     credentialId,
      [out] ByteString credentialSecret,
      [out] NodeId[]   grantedRoles
    );
```

Argument	Description
requestId	The identifier returned from a previous call to *StartRequest*.
cancelRequest	If TRUE, the request is cancelled and no *KeyCredentials* are returned. If FALSE, the normal processing proceeds.
credentialId	The unique identifier for the *KeyCredential*.
credentialSecret	The secret associated with the *KeyCredential*.
certificateThumbprint	The thumbprint of the *Certificate* containing the key used to encrypt the secret. Not specified if the secret is not encrypted.
securityPolicyUri	The *SecurityPolicy* used to encrypt the secret. If not specified, the secret is not encrypted.
grantedRoles	A list of *Roles* which have been granted to *KeyCredential*. If empty, then the information is not relevant or not available.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The *requestId* is does not reference to a valid request for the *Application*.
Bad_RequestNotComplete	The request has not been processed by the *Server* yet.
Bad_UserAccessDenied	The current user does not have the rights required.
Bad_RequestNotAllowed	The *KeyCredential* manager rejected the request. The text associated with the error should indicate the exact reason.

Table 49 specifies the *AddressSpace* representation for the *FinishRequest Method*.

Table 49 – FinishRequest Method AddressSpace definition

Attribute	Value				
BrowseName	FinishRequest				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

8.4.6 Revoke

Revoke is used to revoke a *KeyCredential* used by a *Server*.

This *Method* requires an encrypted channel and that the *Client* provide credentials with administrative rights for the application which is having the credentials revoked.

Signature

```
Revoke (
      [in]  String credentialId
     );
```

Argument	Description
credentialId	The unique identifier for the *KeyCredential*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The *credentialId* is does not reference a valid *KeyCredential*.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 50 specifies the *AddressSpace* representation for the *RevokeKeyCredential Method*.

Table 50 – Revoke Method AddressSpace definition

Attribute	**Value**				
BrowseName	Revoke				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

8.4.7 KeyCredentialAuditEventType

This abstract event is raised when an operation affecting *KeyCredentials* occur.

This *Event* and it subtypes are security related and *Servers* shall only report them to users authorized to view security-related audit events.

Its representation in the *AddressSpace* is formally defined in Table 51.

Table 51 – KeyCredentialAuditEventType definition

Attribute	**Value**				
BrowseName	KeyCredentialAuditEventType				
Namespace	CORE (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *AuditUpdateMethodEventType* defined in IEC 62541-5.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasSubtype	ObjectType	KeyCredentialRequestedAuditEventType		Defined in 8.4.8	
HasSubtype	ObjectType	KeyCredentialDeliveredAuditEventType		Defined in 8.4.9	
HasSubtype	ObjectType	KeyCredentialRevokedAuditEventType		Defined in 8.4.10	
HasSubtype	ObjectType	KeyCredentialUpdatedAuditEventType		Defined in 8.5.6	
HasSubtype	ObjectType	KeyCredentialDeletedAuditEventType		Defined in 8.5.7	

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

The *ResourceUri Property* specifies the URI for the resource which accepts the *KeyCredential.*

8.4.8 KeyCredentialRequestedAuditEventType

This event is raised when a new *KeyCredential* request has been accepted or rejected by the *Server*.

This can be the result of a *StartKeyCredentialRequest Method* call.

Its representation in the *AddressSpace* is formally defined in Table 52.

Table 52 – KeyCredentialRequestedAuditEventType definition

Attribute	Value				
BrowseName	KeyCredentialRequestedAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *KeyCredentialAuditEventType* defined in 8.4.7.					

This EventType inherits all Properties of the KeyCredentialAuditEventType.

8.4.9 KeyCredentialDeliveredAuditEventType

This event is raised when a *KeyCredential* is delivered by the *Server* to an application.

This is the result of a *FinishKeyCredentialRequest Method* completing.

Its representation in the *AddressSpace* is formally defined in Table 53.

Table 53 – KeyCredentialDeliveredAuditEventType definition

Attribute	Value				
BrowseName	KeyCredentialDeliveredAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *KeyCredentialAuditEventType* defined in 8.4.7.					

This EventType inherits all Properties of the KeyCredentialAuditEventType.

8.4.10 KeyCredentialRevokedAuditEventType

This event is raised when a *KeyCredential* is revoked.

This is the result of a *RevokeKeyCredential Method* completing.

Its representation in the *AddressSpace* is formally defined in Table 54.

Table 54 – KeyCredentialRevokedAuditEventType definition

Attribute	Value				
BrowseName	KeyCredentialRevokedAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *KeyCredentialAuditEventType* defined in 8.4.7.					

This EventType inherits all Properties of the KeyCredentialAuditEventType.

8.5 Information Model for push management

8.5.1 General

The *AddressSpace* used for push management is shown in Figure 19. *Clients* interact with the *Nodes* defined in this model when they need update the *KeyCredentials* used by a *Server* to access resources such as *Brokers* or *Authorization Servers*. The *NetworkResources Folder* is a well-known *Object* that appears in the *AddressSpace* of any *Server* which supports *KeyCredential* management.

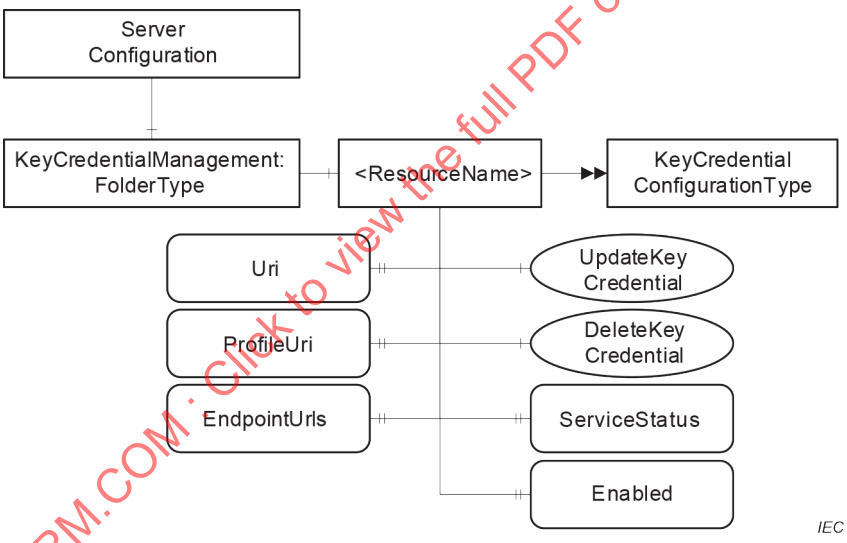


Figure 19 – The AddressSpace used for Push KeyCredential management

8.5.2 KeyCredentialConfiguration

This *Object* is an instance of *FolderType*. It contains The *Objects* which make be accessed via the *Server*. It is the target of an *HasComponent* reference from the *ServerConfiguration Object* defined in 7.7.2. It is defined in Table 55.

Table 55 – KeyCredentialConfiguration Object definition

Attribute	Value				
BrowseName	KeyCredentialConfiguration				
Namespace	CORE (see 3.3)				
TypeDefinition	FolderType defined in IEC 62541-5.				
References	**NodeClass**	**BrowseName**	**TypeDefinition**		**Modelling Rule**
HasComponent	Object	<ServiceName>	KeyCredentialConfigurationType		OptionalPlaceholder

8.5.3 KeyCredentialConfigurationType

This *ObjectType* is the *TypeDefinition* for an *Object* that allows the configuration of *KeyCredentials* used by the *Server*. It also includes basic status information which report problems accessing the resource that might be related to bad *KeyCredentials*. It is defined in Table 56.

Table 56 – KeyCredentialConfigurationType definition

Attribute	Value				
BrowseName	KeyCredentialConfigurationType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory
HasProperty	Variable	ProfileUri	String	PropertyType	Mandatory
HasProperty	Variable	EndpointUrls	String[]	PropertyType	Optional
HasProperty	Variable	ServiceStatus	StatusCode	PropertyType	Optional
HasComponent	Method	UpdateCredential		Defined in 8.5.4	Optional
HasComponent	Method	DeleteCredential		Defined in 8.5.5	Optional

The *ResourceUri Property* uniquely identifies the resource that accepts the *KeyCredentials*.

The *ProfileUri Property* specifies the protocol used to access the resource.

The *EndpointUrls Property* specifies the URLs that the *Server* uses to access the resource.

The *ServiceStatus Property* indicates the result of the last attempt to communicate with the resource. The following common error values are defined:

ServiceStatus	Description
Bad_OutOfService	Communication was not attempted by the *Server* because *Enabled* is FALSE.
Bad_IdentityTokenRejected	Communication failed because the *KeyCredentials* are not valid.
Bad_NoCommunication	Communication failed because the endpoint is not reachable. Where possible a more specific error code should be used. See IEC 62541-4 for a complete list of standard *StatusCodes*.

The UpdateKeyCredential Method is used to change the KeyCredentials used by the Server.

The DeleteKeyCredential Method is used to delete the KeyCredentials stored by the Server.

8.5.4 UpdateCredential

UpdateCredential is used to update a *KeyCredential* used by a *Server*.

The *KeyCredential* secret may be encrypted with the public key of the *Server's Certificate*. The *SecurityPolicyUri* species the algorithm used for encryption. The format of the encrypted data is described in 8.4.5.

This *Method* requires an encrypted channel and that the *Client* provides credentials with administrative rights on the *Server*.

Signature

```
UpdateCredential(
      [in] String     credentialId,
      [in] ByteString credentialSecret,
      [in] String     certificateThumbprint,
      [in] String     securityPolicyUri
     );
```

Argument	Description
credentialId	The unique identifier associated with the *KeyCredential*.
credentialSecret	The secret associated with the *KeyCredential*.
certificateThumbprint	The thumbprint of the *Certificate* used to encrypt the secret. For RSA *SecurityPolicies*, this shall be one of the *ApplicationInstance Certificates* assigned to the *Server*. For ECC *SecurityPolicies*, this field is not specified. Not specified if the secret is not encrypted.
securityPolicyUri	The *SecurityPolicy* used to encrypt the secret. If not specified the secret is not encrypted.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_InvalidArgument	The credentialId or credentialSecret is not valid.
Bad_CertificateInvalid	The *Certificate* is invalid or it is not one of the *Server's Certificates*.
Bad_SecurityPolicyRejected	The *SecurityPolicy* is unrecognized or not allowed.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 57 specifies the *AddressSpace* representation for the *UpdateKeyCredential Method*.

Table 57 – UpdateCredential Method AddressSpace definition

Attribute	Value				
BrowseName	UpdateCredential				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

8.5.5 DeleteCredential

DeleteCredential is used to delete a *KeyCredential* used by a *Server*.

This *Method* requires an encrypted channel and that the *Client* provides credentials with administrative rights on the *Server*.

Signature

```
DeleteCredential()
```

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 58 specifies the *AddressSpace* representation for the *DeleteKeyCredential Method*.

Table 58 – DeleteCredential Method AddressSpace definition

Attribute	Value				
BrowseName	DeleteCredential				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

8.5.6 KeyCredentialUpdatedAuditEventType

This event is raised when a *KeyCredential* is updated.

This *Event* and its subtypes report sensitive security related information. Servers shall only report these *Events* to Clients that are authorized to view such information.

This is the result of a *UpdateCredential Method* completing.

Its representation in the *AddressSpace* is formally defined in Table 59.

Table 59 – KeyCredentialUpdatedAuditEventType definition

Attribute	Value				
BrowseName	KeyCredentialUpdatedAuditEventType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *KeyCredentialAuditEventType* defined in 8.4.7.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *KeyCredentialAuditEventType*.

8.5.7 KeyCredentialDeletedAuditEventType

This event is raised when a *KeyCredential* is updated.

This is the result of a *DeleteCredential Method* completing.

Its representation in the *AddressSpace* is formally defined in Table 60.

Table 60 – KeyCredentialUpdatedAuditEventType definition

Attribute	Value				
BrowseName	KeyCredentialDeletedAuditEventType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *KeyCredentialAuditEventType* defined in 8.4.7.					
HasProperty	Variable	ResourceUri	String	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *KeyCredentialAuditEventType*.

9 Authorization Services

9.1 Overview

Authorization Services provide *Access Tokens* to *Clients* that may use them to access resources. A *Server,* such as a GDS, with *Authorization Service* capabilities may support one or more *AuthorizationService Objects* (see 9.5.2) which may represent an internal *Authorization Service* or be an API to an external *Authorization Service*. The *Authorization Service* is best used in conjunction with the *Role* model defined in IEC 62541-5. In this scenario, the mapping rules assigned to the *Roles* known to the *Server* are used to populate an *Access Token* with the *Roles* associated with the *UserIdentity* provided when the *Client* submits the request. This scenario is illustrated in Figure 20.

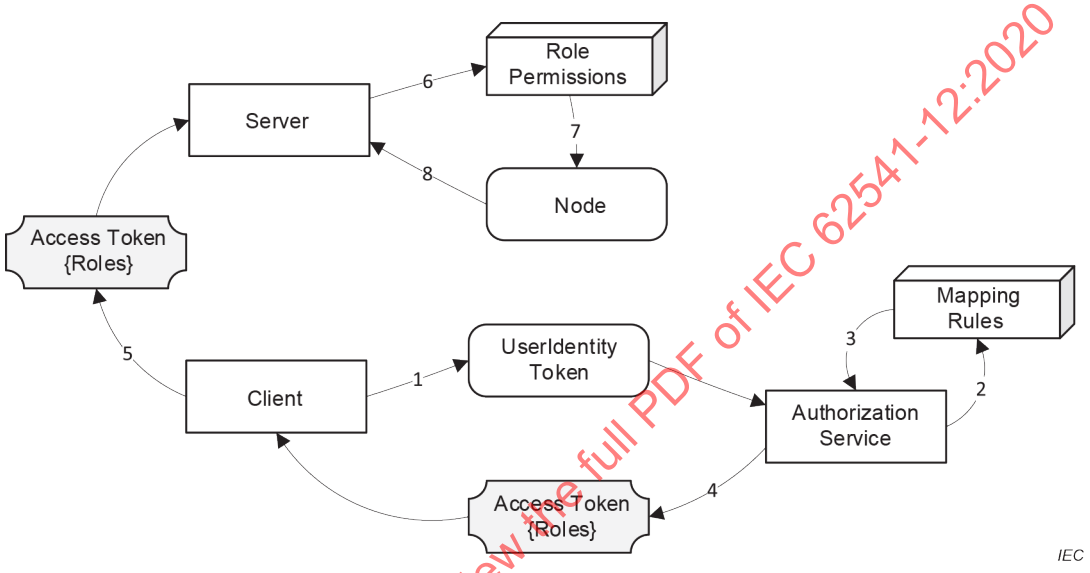


Figure 20 – Roles and Authorization Services

When requesting *Access Tokens* from an *AuthorizationService Object* there are three primary use cases based on where the *UserIdentityToken* comes from: Implicit, Explicit and Chained. These use cases are discussed below. The Implicit and Explicit use cases are implementations of the 'Indirect' model for *Authorization Services* described in IEC 62541-4. The Chained use case is an implementation of the "Direct" model.

9.2 Implicit

The implicit use case means the *Client's Application Certificate* and any *UserIdentityToken* associated with the *Session* is used to determine whether an *Access Token* is permitted and what claims are available. This use case is illustrated in Figure 21.

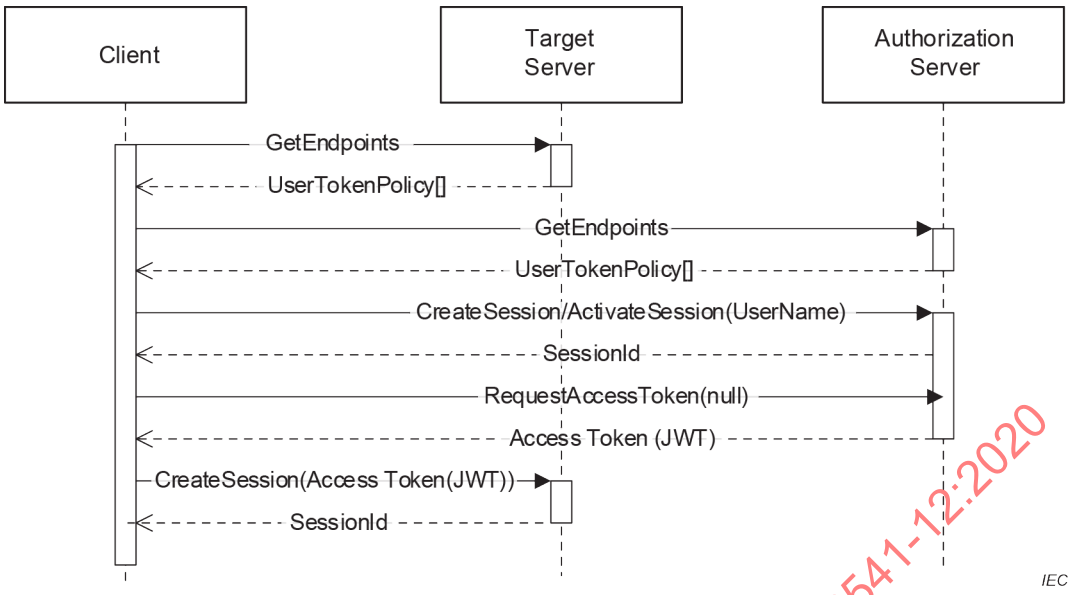


Figure 21 – Implicit authorization

The Target *Server* is the *Server* that the *Client* wishes to access. It publishes a *UserTokenPolicy* that indicates that it accepts *Access Tokens* from an Authorization Server at a URL specified in the policy. The policy also contains the *NodeId* of the *AuthorizationService Object* which is used to request the *Access Token*.

The *Client* needs to be trusted by the Authorization Server and this could require the *Client* to present user credentials. These credentials can be provided to the *Client* out-of-band (e.g. an administrator specified them in the *Client* configuration file).

The *Session* may be created explicitly with a call to *CreateSession* or it can be implicit via a *Session*-less *Method Call*.

After creating the *Session,* the *Client* calls the *RequestAccessToken Method* on the *AuthorizationService Object*. The Authorization Server determines if the *Client* is permitted to receive an *Access Token* and populates it with any claims granted to the *Client*. This claims may include *Roles* granted to the *Session* by applying the mapping rules for the Roles (see IEC 62541-3).

Once the *Client* has the *Access Token,* it passes the *Access Token* to the Target *Server* which validates the *Access Token*, as described in IEC 62541-4. The Target *Server* is configured out-of-band with the *Certificate* needed to validate the *Access Tokens* issued by the Authorization *Server*.

9.3 Explicit

The explicit use case means the *Client* provides the *UserIdentityToken* used to determine whether an *Access Token* is permitted and what claims are available in the call to *RequestAccessToken*. This use case is illustrated in Figure 22.

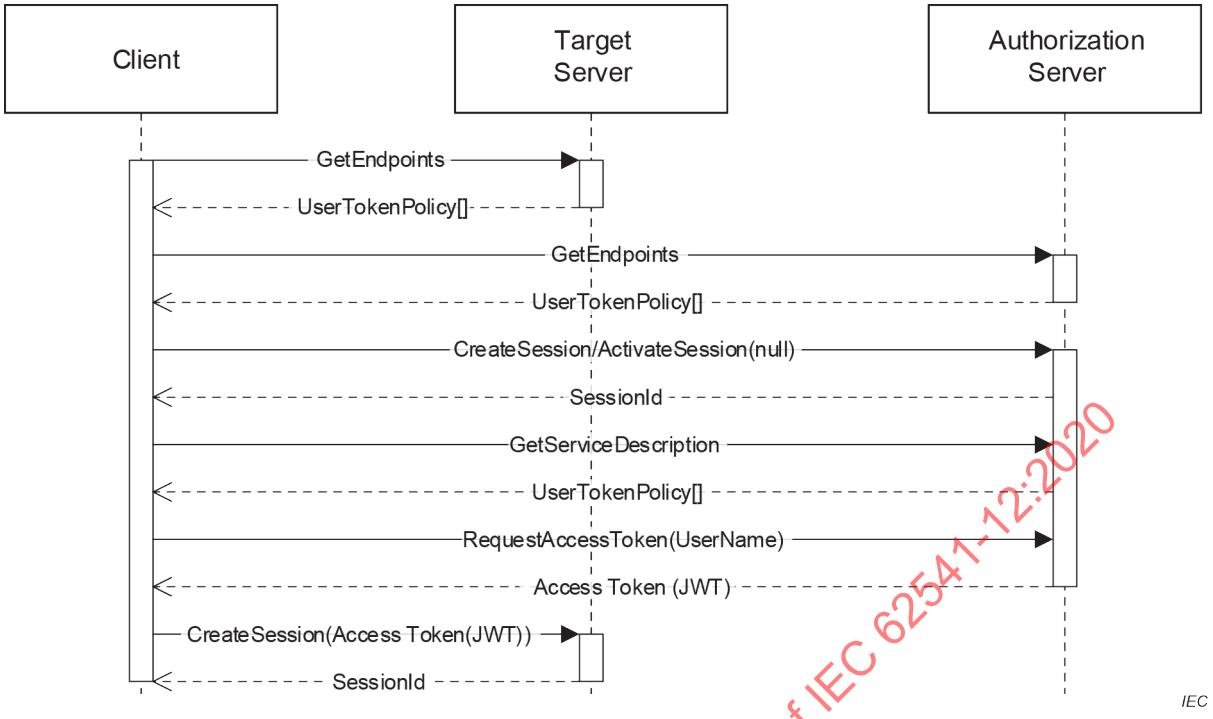


Figure 22 – Explicit authorization

The Target *Server* is the *Server* that the *Client* wishes to access. The initial interactions are the same as with the Implicit use case described in 9.2.

The *Session* may be created explicitly with a call to *CreateSession* or it can be implicit via a *Session*-less *Method Call*.

After creating the *Session*, the *Client* reads the available *UserTokenPolicies* from the *AuthorizationService Object* if it has not previously cached the information. It then chooses one that matches credentials that it has been provided out-of-band. The *Client* then calls the *RequestAccessToken Method* on the *AuthorizationService Object*.

The Authorization *Server* determines if the *Client* is permitted to receive an *Access Token*. The rest of the interactions are the same as described in 9.2.

9.4 Chained

The chained use case means the *Client* provides an *Access Token* issued by another *Authorization Service* acting as an *Identity Provider*. This use case is illustrated in Figure 23.

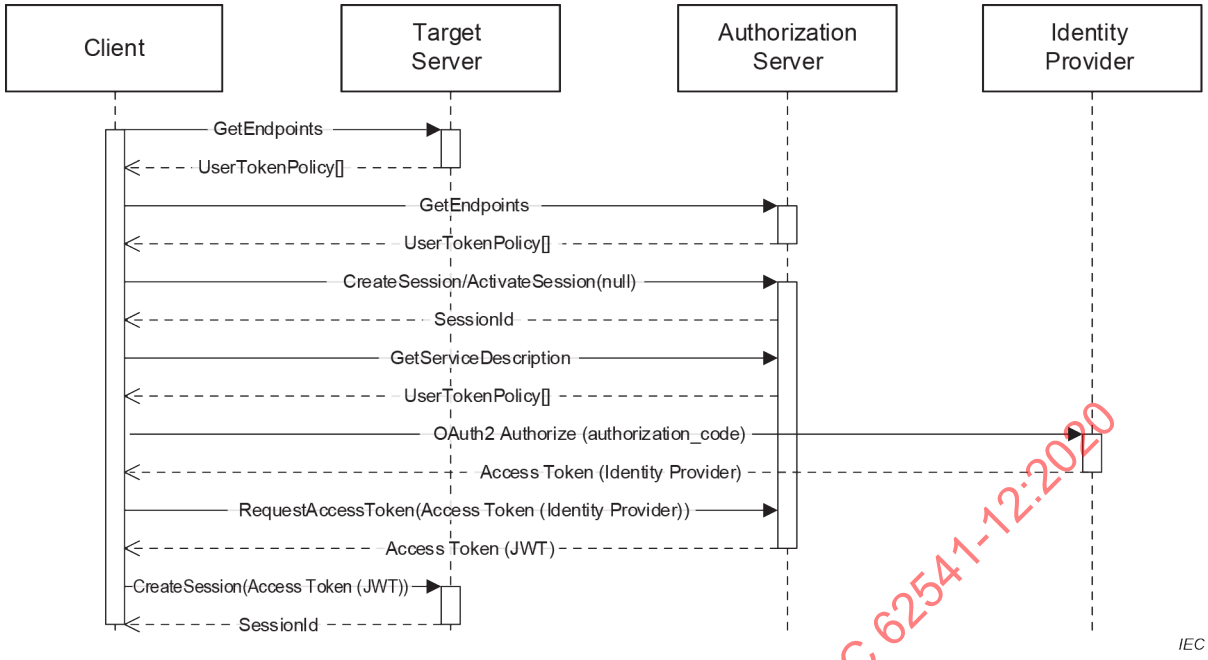


Figure 23 – Chained authorization

The Target *Server* is the *Server* that the *Client* wishes to access. The initial interactions are the same as with the Implicit use case described in 9.2.

The *Session* may be created explicitly with a call to *CreateSession* or it can be implicit via a *Session*-less *Method Call*.

After creating the *Session*, the *Client* reads the available *UserTokenPolicies* from the *AuthorizationService Object* if it has not previously cached the information. It then chooses one that references an *Identity Provider* for the user identities that it has available. The user identities may be provided out-of-band or they may be provided by an interactive user. The *Client* then requests an *Access Token* from the *Identity Provider*.

The *Client* then calls the *RequestAccessToken Method* on the *AuthorizationService Object* and passes the *Access Token* from the *Identity Provider*.

The Authorization *Server* determines if the *Client* is permitted to receive an *Access Token* based on the claims granted by the *Identity Provider*. The rest of the interactions are the same as described in 9.2.

9.5 Information Model for Requesting Access Tokens

9.5.1 Overview

The information model for *Authorization Services* which allow *Clients* to request *Access Tokens* from a *Server* is shown in Figure 24.

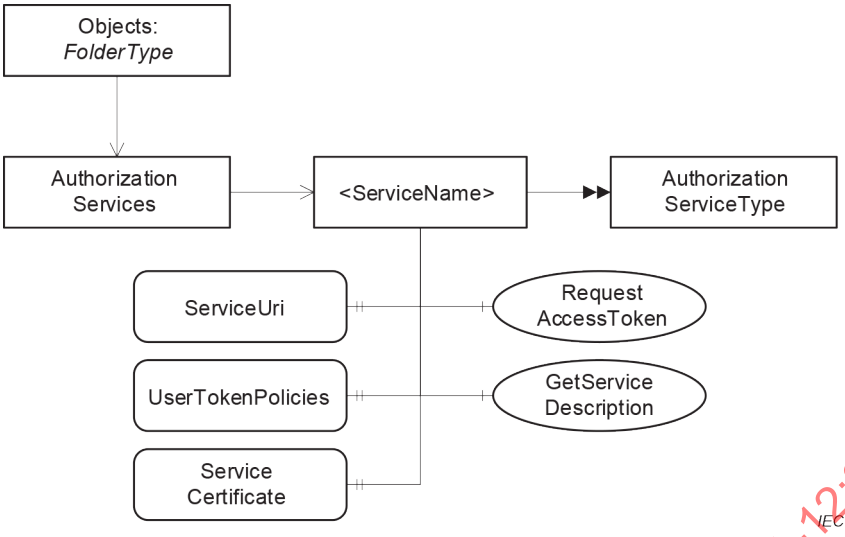


Figure 24 – The Model for Requesting Access Tokens from Authorization Services

9.5.2 AuthorizationServices

This *Object* is an instance of *FolderType*. It contains The *AuthorizationService Objects* which may be accessed via the GDS. It is the target of an *Organizes* reference from the *Objects Folder* defined in IEC 62541-5. It is defined in Table 61.

Table 61 – AuthorizationServices Object definition

Attribute	Value			
BrowseName	AuthorizationServices			
Namespace	GDS (see 3.3)			
TypeDefinition	*FolderType* defined in IEC 62541-5.			
References	**NodeClass**	**BrowseName**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<ServiceName>	AuthorizationServiceType	OptionalPlaceholder

9.5.3 AuthorizationServiceType

This *ObjectType* is the *TypeDefinition* for an *Object* that allows access to an *Authorization Service*. It is defined in Table 62.

Table 62 – AuthorizationServiceType definition

Attribute	Value				
BrowseName	AuthorizationServiceType				
Namespace	GDS (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasProperty	Variable	ServiceUri	String	PropertyType	Mandatory
HasProperty	Variable	ServiceCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	UserTokenPolicies	UserTokenPolicy []	PropertyType	Optional
HasComponent	Method	GetServiceDescription		Defined in 9.5.5.	Mandatory
HasComponent	Method	RequestAccessToken		Defined in 9.5.4.	Optional

The *ServiceUri* is a globally unique identifier that allows a *Client* to correlate an instance of *AuthorizationServiceType* with instances of *AuthorizationServiceConfigurationType* (see 9.6.3).

The *ServiceCertificate* is the complete chain of *Certificates* needed to validate the *Access Tokens* (see IEC 62541-6 for information on encoding chains).

The *UserTokenPolicies Property* specifies the *UserIdentityTokens* which are accepted by the *RequestAccessToken Method.*

The *GetServiceDescription Method* is used read the metadata needed to request *Access Tokens.*

The *RequestAccessToken Method* is used to request an *Access Token* from the *Authorization Service.*

9.5.4 RequestAccessToken

RequestAccessToken is used to request an *Access Token* from an *Authorization Service*. The scenarios where this this *Method* is used are described fully in 9.2, 9.3 and 9.4.

The *PolicyId* and *UserTokenType* of the *identityToken* shall match one of the elements of the *UserTokenPolicies Property*. If the *identityToken* is not provided the *Server* should use the *ApplicationInstanceCertificate* and/or the *UserIdentityToken* provided for the *Session* (or the request if using a *Session*-less *Method Call*) to determine privileges.

If the associated *UserTokenPolicy* provides a *SecurityPolicyUri*, then the *identityToken* is encrypted and digitally signed using the format defined for *UserIdentityToken* secrets in IEC 62541-4.

For *UserNameIdentityTokens*, the secret is the password and the signature is created with the *Client ApplicationInstanceCertificate*. The signed and encrypted secret is passed in the *password* field.

For *X.509 v3IdentityTokens*, the secret is null, and the signature is created with the key associated with user *Certificate*. The signed and encrypted secret is passed in the *certificateData* field.

For *IssuedIdentityTokens* the secret is the token and the signature is created with the key associated a user *Certificate* or the *Client ApplicationInstanceCertificate.* The signed and encrypted secret is passed in the *tokenData* field.

The *Server* shall check the *signingTime* in against the current system clock. The *Server* shall reject the request if the *signingTime* is outside of a configurable range. A suitable default value is 5 min. The permitted clock skew is a *Server* configuration parameter.

This *Method* requires an encrypted channel and that the *Client* provides credentials with administrative rights for the application which is having the credentials revoked.

Signature

```
RequestAccessToken (
      [in]  UserIdentityToken identityToken,
      [in]  String resourceId,
      [out] String accessToken
    );
```

Argument	Description
identityToken	The identity used to authorize the *Access Token* request.
resourceId	The identifier for the Resource that the *Access Token* is used to access. This is usually the *ApplicationUri* for a *Server*.
accessToken	The *Access Token* granted to the application.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_IdentityTokenInvalid	The *identityToken* does not match one of the allowed *UserTokenPolicies*.
Bad_IdentityTokenRejected	The *identityToken* was rejected.
Bad_NotFound	The *resourceId* is not known to the *Server*.
Bad_UserAccessDenied	The current user does not have the rights required.

Table 63 specifies the *AddressSpace* representation for the *RequestAccessToken Method*.

Table 63 – RequestAccessToken Method AddressSpace definition

Attribute	Value				
BrowseName	RequestAccessToken				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.5.5 GetServiceDescription

GetServiceDescription is used to read the metadata needed to request *Access Tokens* from the *Authorization Service*.

Signature

```
GetServiceDescription (
      [out] String serviceUri
      [out] ByteString serviceCertificate
      [out] UserTokenPolicy[] policies
    );
```

Argument	Description
serviceUri	A globally unique identifier for the *Authorization Service*.
serviceCertificate	The complete chain of *Certificates* needed to validate the *Access Tokens* provided by the *Authorization Service*.
policies	The *UserIdentityTokens* accepted by the *Authorization Service*.

Method Result Codes (defined in Call Service)

Result Code	Description
Bad_UserAccessDenied	The current user does not have the rights required.

Table 64 specifies the *AddressSpace* representation for the *GetServiceDescription Method*.

Table 64 – GetServiceDescription Method AddressSpace definition

Attribute	Value				
BrowseName	GetServiceDescription				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.5.6 AccessTokenIssuedAuditEventType

This event is raised when a *AccessToken* is issued.

This is the result of a *RequestAccessToken Method* completing.

This *Event* and it subtypes are security related and *Servers* shall only report them to users authorized to view security-related audit events.

Its representation in the *AddressSpace* is formally defined in Table 65.

Table 65 – AccessTokenIssuedAuditEventType definition

Attribute	Value				
BrowseName	AccessTokenIssuedAuditEventType				
Namespace	GDS (see 3.3)				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the AuditUpdateMethodEventType defined in IEC 62541-5.					

This *EventType* inherits all *Properties* of the *AuditUpdateMethodEventType*. Their semantic is defined in IEC 62541-5.

9.6 Information Model for configuring Servers

9.6.1 Overview

The information model used to provide *Servers* with the information needed to accept *Access Tokens* from *Authorization Services* in Figure 25.

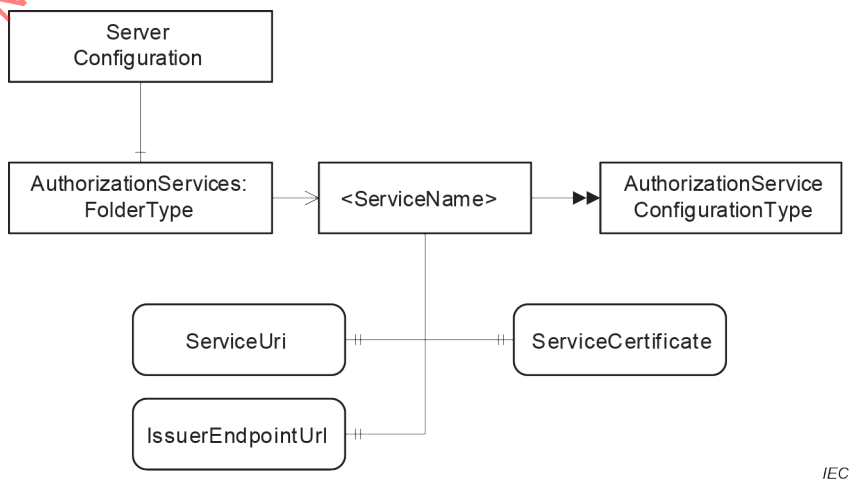


Figure 25 – The Model for configuring Servers to use Authorization Services

If a *Server* is also a *Client* that needs to access the *Authorization Service,* the necessary *KeyCredentials* can be provided with the push configuration management model (see 8.3).

9.6.2 AuthorizationServices

This *Object* is an instance of *FolderType.* It contains The *AuthorizationServiceConfiguration Objects* which may be accessed via the *Server*. It is the target of an *HasComponent* reference from the *ServerConfiguration Object* defined in 7.7.2. It is defined in Table 66.

Table 66 – AuthorizationServices Object definition

Attribute	Value			
BrowseName	AuthorizationServices			
Namespace	CORE (see 3.3)			
TypeDefinition	*FolderType* defined in IEC 62541-5.			
References	**NodeClass**	**BrowseName**	**TypeDefinition**	**Modelling Rule**
HasComponent	Object	<ServiceName>	AuthorizationServiceConfigurationType	OptionalPlaceholder

9.6.3 AuthorizationServiceConfigurationType

This *ObjectType* is the *TypeDefinition* for an *Object* that allows the configuration of an *Authorization Service* used by a *Server*. It is defined in Table 67.

Table 67 – AuthorizationServiceConfigurationType definition

Attribute	Value				
BrowseName	AuthorizationServiceConfigurationType				
Namespace	CORE (see 3.3)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseObjectType* defined in IEC 62541-5.					
HasProperty	Variable	ServiceUri	String	PropertyType	Mandatory
HasProperty	Variable	ServiceCertificate	ByteString	PropertyType	Mandatory
HasProperty	Variable	IssuerEndpointUrl	String	PropertyType	Mandatory

The *ServiceUri Property* uniquely identifies the *Authorization Service*.

The *ServiceCertificate Property* has the *Certificate(s)* needed to verify *Access Tokens* issued by the *Authorization Service*. The value is the complete chain of Certificate needed for verification (see IEC 62541-6 for information on encoding chains).

The *IssuerEndpointUrl* is the value of the *IssuerEndpointUrl* in *UserTokenPolicies* which require the use of the Authorization Service. These contents of the field depend on the Authorization Service and are described in IEC 62541-6.

Annex A
(informative)

Deployment and configuration

A.1 Firewalls and discovery

Many systems will have multiple networks that are isolated by firewalls. These firewalls will frequently hide the network addresses of the hosts behind them unless the Administrator has specifically configured the firewall to allow external access. In some networks, the Administrator will place hosts with externally available *Servers* outside the firewall as shown in Figure A.1.

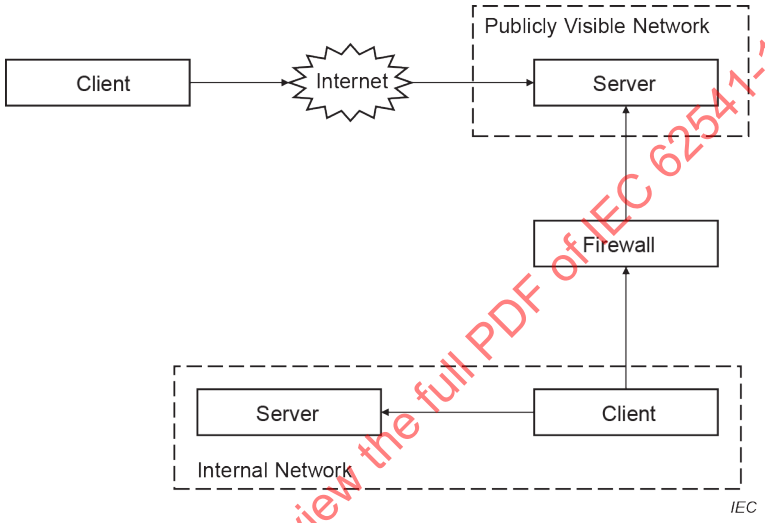


Figure A.1 – Discovering Servers outside a firewall

In this configuration, *Servers* running on the publicly visible network will have the same network address from the perspective of all *Clients* which means the URLs returned by *DiscoveryServers* are not affected by the location of the *Client*.

In other networks, the Administrator will configure the firewall to allow access to selected *Servers*. An example is shown in Figure A.2.

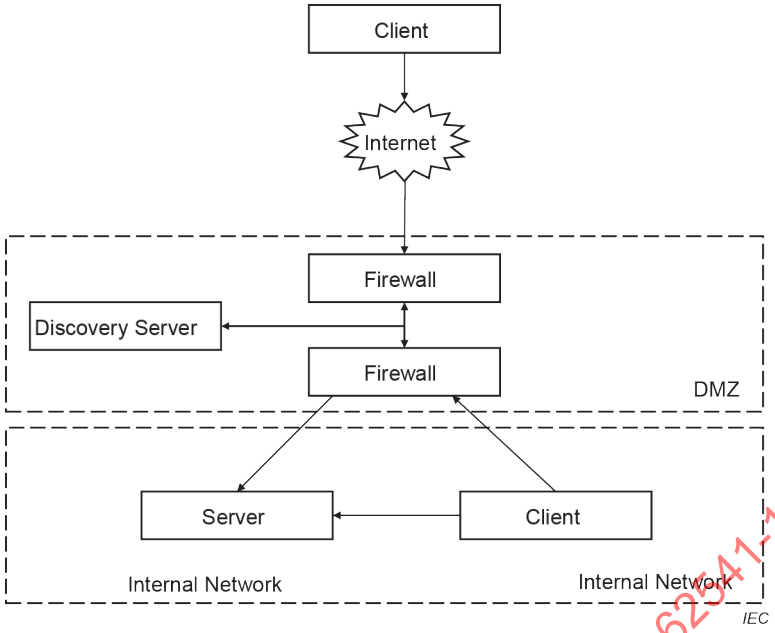


Figure A.2 – Discovering Servers behind a firewall

In this configuration, the address of the *Server* that the Internet *Client* sees will be different from the address that the Internet *Client* sees. This means that the *Server's DiscoveryEndpoint* would return incorrect URLs to the Internet *Client* (assuming it was configured to provide the internal URLs).

Administrators can correct this problem by configuring the *Server* to use multiple *HostNames*. A *Server* that has multiple *HostNames* shall look at the *EndpointUrl* passed to the *GetEndpoints* or *CreateSession* services and return *EndpointDescriptions* with URLs that use the same *HostName*. A Server with multiple *HostNames* shall also return an *Application Instance Certificate* that specifies the *HostName* used in the URL it returns. An Administrator may create a single *Certificate* with multiple HostNames or assign different *Certificates for each HostName* that the *Server* supports.

Note that *Servers* may not be aware of all *HostNames* which can be used to access the *Server* (i.e. a NAT firewall) so *Clients* need to handle the case where the URL used to access the *Server* is different from the *HostNames* in the *Certificate*. This is discussed in more detail in IEC 62541-4.

Administrators may also wish to set up a *DiscoveryServer* that is configured with the *ApplicationDescriptions* for *Servers* that are accessible to external *Clients*. This *DiscoveryServer* would have to substitute its own *Endpoint* for the *DiscoveryUrls* in all *ApplicationDescriptions* that it returns when a *Client* calls *FindServers*. This would tell the *Client* to call the *DiscoveryServer* back when it wishes to connect to the *Server*. The *DiscoveryServer* would then request the *EndpointDescriptions* from the actual Server as shown in Figure A.3. At this point the *Client* would have all the information it needs to establish a secure channel with the *Server* behind the firewall.

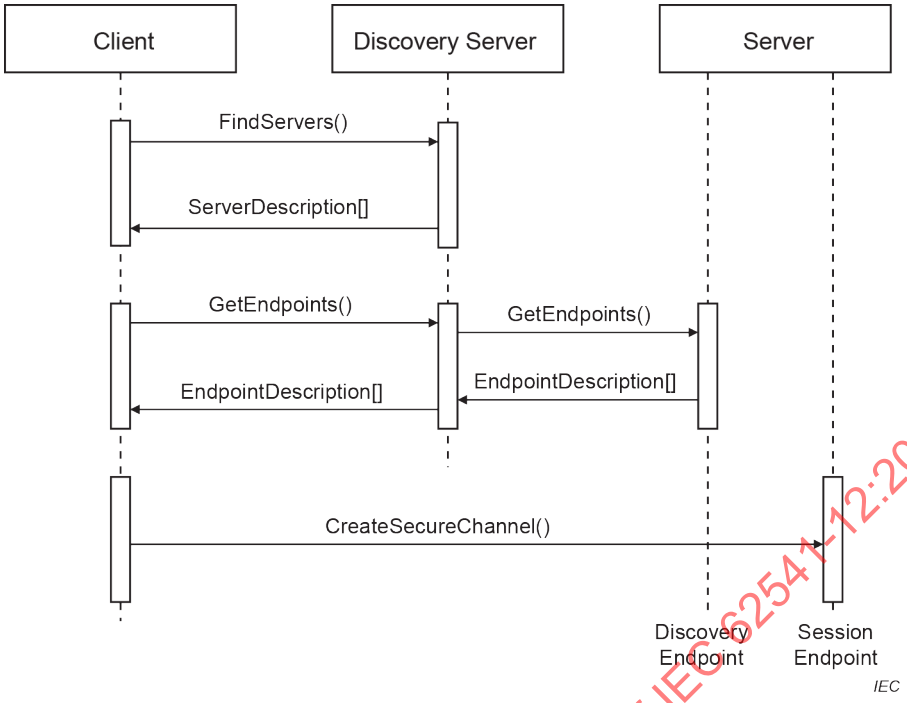


Figure A.3 – Using a Discovery Server with a firewall

In this example, the *DiscoveryServer* outside of the firewall allows the *Administrator* to close off the *Server's DiscoveryEndpoints* to every Client other than the *DiscoveryServer*. The Administrator could eliminate that hole as well if it stored the *EndpointDescriptions* on the *DiscoveryServer*. This allows an Administrator to configure a system in which no public access is allowed to any application behind the firewall. The only access behind the firewall is via a secure connection.

The *DiscoveryServer* could also be replaced with a *DirectoryService* that stores the *ApplicationDescriptions* and/or the *EndpointDescriptions* for the *Servers* behind the firewalls.

A.2 Resolving references to remote Servers

The UA *AddressSpace* supports references between *Nodes* that exist in different *Server AddressSpace* spaces. These references are specified with an *ExpandedNodeId* that includes the URI of the *Server* which owns the *Node*. A *Client* that wishes to follow a reference to an external *Node* should map the *ApplicationUri* onto an *EndpointUrl* that it can use. A *Client* can do this by using the *GlobalDiscoveryServer* that knows about the *Server*. The process of connecting to a *Server* containing a remote *Node* is illustrated in Figure A.4.

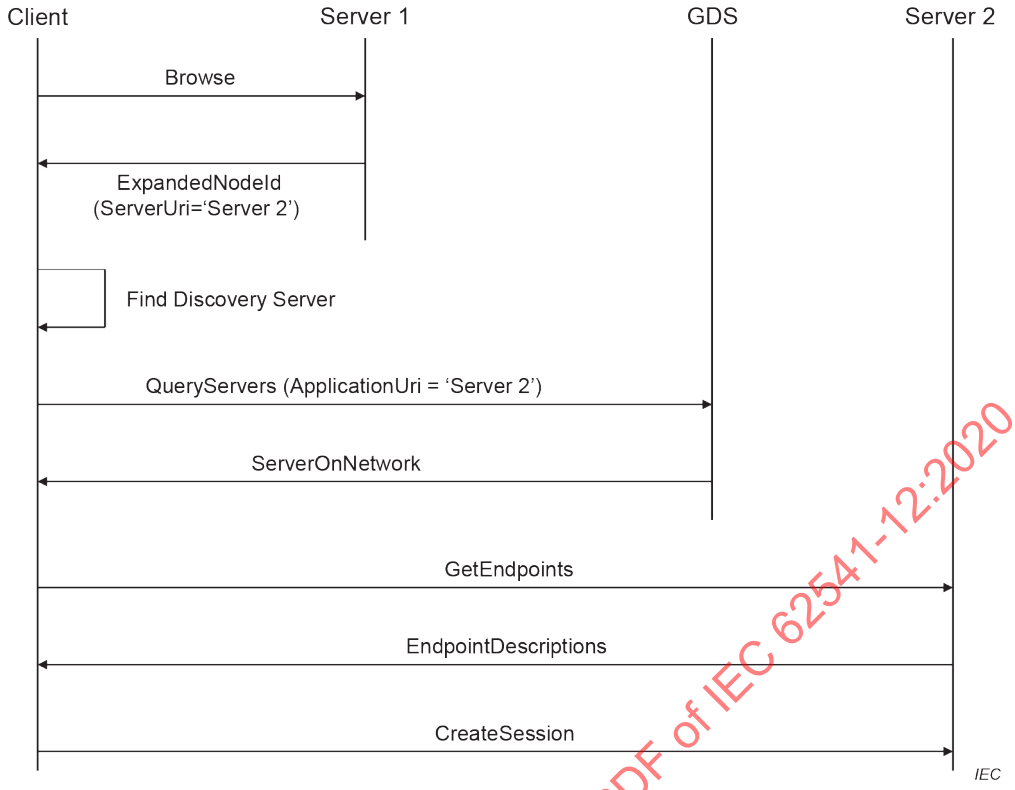


Figure A.4 – Following References to Remote Servers

If a GDS not available, *Client* may use other strategies to find the *Server* associated with the URI.

Annex B
(normative)

Constants

This document defines *Nodes* that are part of the base OPC UA Specification. The numeric identifiers for these *Nodes* are part of the complete list of identifiers defined in IEC 62541-6.

In addition, this document defines *Nodes* which are only used by *GlobalDiscoveryServers*.

The *NamespaceUri* for any GDS specific *NodeIds* is http://opcfoundation.org/UA/GDS/

The CSV released with this version of the standards can be found here:

http://www.opcfoundation.org/UA/schemas/1.04/Opc.Ua.Gds.NodeIds.csv

NOTE The latest CSV that is compatible with this version of the standard can be found here:

http://www.opcfoundation.org/UA/schemas//Opc.Ua.Gds.NodeIds.csv

Annex C
(normative)

OPC UA Mapping to mDNS

C.1 DNS Server (SRV) record syntax

Annex C describes the OPC UA specific requirements which are above and beyond the more general requirements of the *mDNS* specification.

mDNS uses DNS SRV records to advertise the services (a.k.a. the *DiscoveryUrls* for the *Servers*) available on the network.

An SRV record has the form:

```
_service._proto.name TTL class SRV priority weight port target
```

service: the symbolic name of the desired service. For OPC UA this field shall be one of service names for OPC UA which are defined in Table C.1.

Table C.1 – Allowed mDNS service names

Service Name	Description
_opcua-tcp	The *DiscoveryUrl* supports the OPC UA TCP mapping (see IEC 62541-6). This name is assigned by IANA.
_opcua-tls	The *DiscoveryUrl* supports the OPC UA WebSockets mapping (see IEC 62541-6). Note that WebSockets mapping supports multiple encodings. If a *Client* supports more than one encoding, it should attempt to use the alternate encodings if an error occurs during connect. This name is assigned by IANA.

proto: the transport protocol of the desired service; for OPC UA, this field shall be '_tcp'.

The other fields have no OPC UA specific requirements.

An example SRV record in textual form that might be found in a zone file might be the following:

```
_opcua-tcp._tcp.example.com.     86400     IN     SRV     0     5     4840
uaserver.example.com.
```

This points to a server named `uaserver.example.com` listening on TCP port 4840 for OPC UA TCP requests. The priority given here is 0, and the weight is 5 (the priority and weights are not important for OPC UA). The *mDNS* specification describes the rest of the fields in detail.

C.2 DNS Text (TXT) record syntax

The SRV record has a TXT record associated with it that provides additional information about the *DiscoveryUrl*. The format of this record is a sequence of strings prefixed by a length. This specification adopts the key-value syntax for TXT records described in *DNS-SD*.

Table C.2 defines the syntax for strings that may in the TXT record.

Table C.2 – DNS TXT record string format

Key-Value Format	Description
path=/<path>	Specifies the text that appears after the port number when constructing a URL. This text always starts with a forward slash (/).
caps=<capability1>,<capability2>	Specifies the capabilities supported by the *Server*. These are short (≤ 8 characters) strings which are published by the OPC Foundation (see Annex D). The number of capabilities supported by a Server should be less than 10.

The *MulticastExtension* shall convert *DiscoveryUrls* to and from these SRV records.

C.3 DiscoveryUrl mapping

An *DiscoveryUrl* has the form:

```
scheme://hostname:port/path
```

scheme: the protocol used to establish a connection.

hostname: the domain name or *IPAddress* of the host where the *Server* is running.

port: the TCP port on which the *Server* is to be found.

path: additional data used to identify a specific *Server*.

Table C.3 – DiscoveryUrl to DNS SRV and TXT Record Mapping

URL Field	Mapping
scheme	The scheme maps onto SRV record service field. The following mappings are defined at this time: opc.tcp → _opcua-tcp._tcp. opc.wss → _opcua-tls._tcp. https → _opcua-https._tcp. The first two are OPC UA service names assigned by IANA. Additional service names may be added in the future. The endpoint shall support the default transport profile for the scheme.
hostname	The hostname maps onto the SRV record target field. If the hostname is an *IPAddress* then it shall be converted to a domain name. If this cannot be done then LDS shall report an error.
port	The port maps onto the SRV record port field.
path	The path maps onto the path string in the TXT record (see Table C.2).

Suitable default values should be chosen for fields in a SRV record that do not have a mapping specified in Table C.3. E.g. TTL = 86 400, class = IN, priority = 0, weight = 5.

Annex D
(normative)

Server Capability Identifiers

Clients benefit if they have more information about a *Server* before they connect, however, providing this information imposes a burden on the mechanisms used to discover *Servers*. The challenge is to find the right balance between the two objectives.

ServerCapabilityIdentifiers are the way this specification achieves the balance. These identifiers are short and map onto a subset of OPC UA features which are likely to be useful during the discovery process. The identifiers are short because of length restrictions for fields used in the mDNS specification. Table D.1 is a non-normative list of possible identifiers.

Table D.1 – Examples of *ServerCapabilityIdentifiers*

Identifier	Description
NA	No capability information is available. Cannot be used in combination with any other capability.
DA	Provides current data.
HD	Provides historical data.
AC	Provides alarms and conditions that may require operator interaction.
HE	Provides historical alarms and events.
GDS	Supports the Global Discovery Server information model.
LDS	Only supports the Discovery Services. Cannot be used in combination with any other capability.
DI	Supports the Device Integration (DI) information model (see DI).
ADI	Supports the Analyser Device Integration (ADI) information model (see ADI).
FDI	Supports the Field Device Integration (FDI) information model (see FDI).
FDIC	Supports the Field Device Integration (FDI) Communication Server information model (see FDI).
PLC	Supports the PLCopen information model (see PLCopen).
S95	Supports the ISA95 information model (see ISA-95).
RCP	Supports the reverse connect capabilities defined in IEC 62541-6.
PUB	Supports the *Publisher* capabilities defined in IEC 62541-14.

The normative set of *ServerCapabilityIdentifiers* can be found here:

http://www.opcfoundation.org/UA/schemas/1.04/ServerCapabilities.csv

Application developers shall always use the linked CSV.

Applications that support the PUB capability can send PubSub Messages but may not support the PubSub information model.

Client applications that support the RCP capability allow *Servers* to connect, however, they do not support *GetEndpoints Service.*

Annex E
(normative)

DirectoryServices

E.1 Global Discovery via other directory services

Many organizations will deploy *DirectoryServices* such as LDAP or UDDI to manage resources available on their network. A *Client* can use these services as a way to find *Servers* by using APIs specific to *DirectoryService* to query for UA *Servers* or UA *DiscoveryServers* available on the network. The *Client* would then use the URLs for *DiscoveryEndpoints* stored in the *DirectoryService* to request the *EndpointDescriptions* necessary to connect to an individual *Server*.

Some implementations of a *GlobalDiscoveryServer* will be a front-end for a standard *Directory Service*. In these cases, the *QueryServers* method will return the same information as the *DirectoryService* API. The discovery process for this scenario is illustrated in Figure E.1.

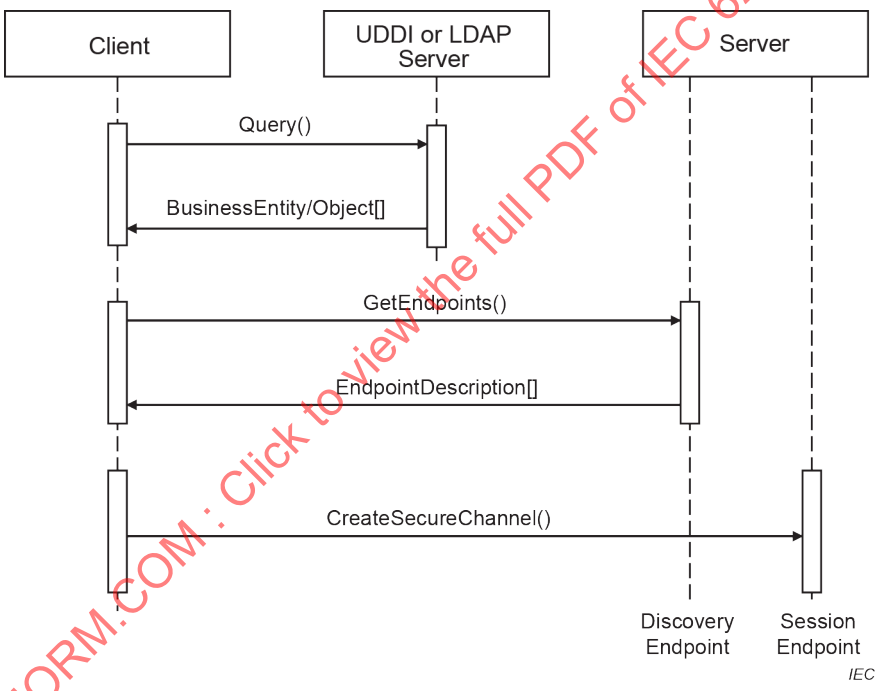


Figure E.1 – The UDDI or LDAP Discovery process

E.2 UDDI

UDDI registries contain *businessEntities* which provide one or more *businessServices*. The *businessServices* have one or more *bindingTemplates*. *bindingTemplates* specify a physical address and a *Server* Interface (called a tModel). Figure E.2 illustrates the relationships between the UDDI registry elements.

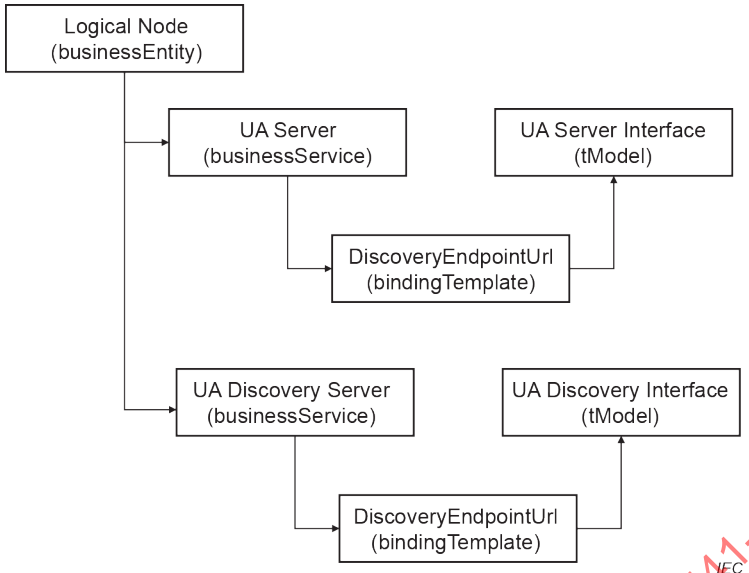


Figure E.2 – UDDI registry structure

This specification defines standard tModels which shall be referenced by businessServices that support UA. The standard UA tModels are shown in Table E.1.

Table E.1 – UDDI tModels

Name	domainKey	uuidKey
Server	uddi:server.ua.opcfoundation.org	uddi:AA206B41-EC9E-49a4-B789-4478C74120B5
DiscoveryServer	uddi:discoveryserver.ua.opcfoundation.org	uddi:AA206B42-EC9E-49a4-B789-4478C74120B5

The name of the businessService elements should be the same as the *ApplicationName* for the UA application. The serviceKey shall be the *ApplicationUri*. At least one bindingTemplate shall be present and the accessPoint shall be the URL of the *DiscoveryEndpoint* for the UA server identified by the serviceKey. Servers with multiple *DiscoveryEndpoints* would have multiple bindingTemplates.

A UDDI registry will generally only contain UA servers, however, there are situations where the administrators cannot know what *Servers* are available at any given time and will find it more convenient to place a *DiscoveryServer* in the registry instead.

E.3 LDAP

LDAP servers contain *objects* organized into hierarchies. Each object has an *objectClass* which specifies a number of *attributes*. *Attributes* have values which describe an *object*. Figure E.3 illustrates a sample LDAP hierarchy which contains entries describing UA servers.

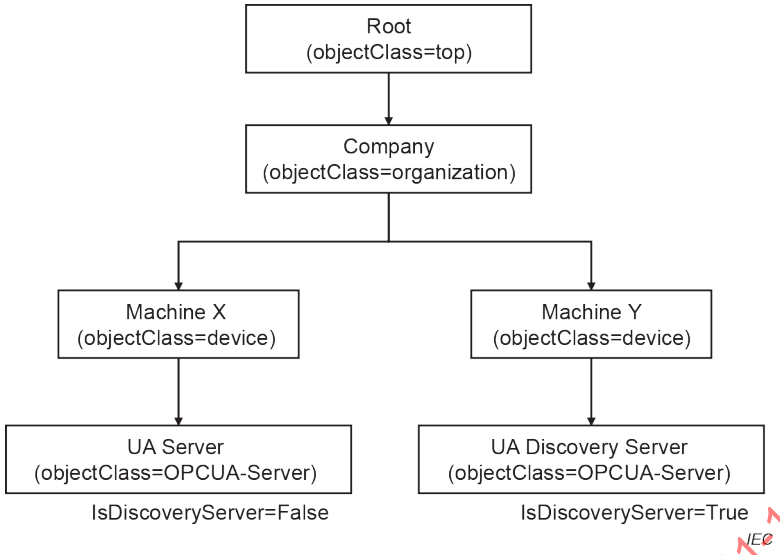


Figure E.3 – Sample LDAP hierarchy

UA applications are stored in LDAP servers as entries with the UA defined objectClasses associated with them. The schema for the objectClasses defined for UA are shown in Table E.2.

Table E.2 – LDAP object class schema

Name	LDAP Name	Type	OID
Application	opcuaApplication	Structural	1.2.840.113556.1.8000.2264.1.12.1
ApplicationName	cn	String (Required)	Built-in
HostName	dNSName	String	Built-in
ApplicationUri	opcuaApplicationUri	Name	1.2.840.113556.1.8000.2264.1.12.1.1
ApplicationType	opcuaApplicationType	Boolean	1.2.840.113556.1.8000.2264.1.12.1.3
DiscoveryUrl	opcuaDiscoveryUrl	String, Multi-valued	1.2.840.113556.1.8000.2264.1.12.1.4

This OID is globally unique and can use used with any LDAP implementation.

Administrators may extend the LDAP schema by adding new attributes.

Annex F
(normative)

Local Discovery Server

F.1 Certificate store directory layout

A recommended directory layout for *Applications* that store their *Certificates* on a file system is shown in Table F.1. The Local Discovery Server shall use this structure.

This structure is based on the rules defined in IEC 62541-6.

Table F.1 – Application Certificate store directory layout

Path	Description
<root>	A descriptive name for the trust list.
<root>/own	The *Certificate* store which contains private keys used by the application.
<root>/own/certs	Contains the X.509 v3 *Certificates* associated with the private keys in the /private directory.
<root>/own/private	Contains the private keys used by the application.
<root>/trusted	The *Certificate* store which contains trusted *Certificates*.
<root>/trusted/certs	Contains the X.509 v3 *Certificates* which are trusted.
<root>/trusted/crl	Contains the X.509 v3 CRLs for any *Certificates* in the /certs directory.
<root>/issuer	The *Certificate* store which contains the CA *Certificates* needed for validation.
<root>/issuer/certs	Contains the X.509 v3 *Certificates* which are needed for validation.
<root>/issuer/crl	Contains the X.509 v3 CRLs for any *Certificates* in the /certs directory.
<root>/rejected	The *Certificate* store which contains certificates which have been rejected.
<root>/rejected/certs	Contains the X.509 v3 *Certificates* which have been rejected.

All X.509 v3 certificates are stored in DER format and have a ".der" extension on the file name.

All CRLs are stored in DER format and have a ".crl" extension on the file name.

Private keys should be in PKCS #12 format with a ".pfx" extension or in the OpenSSL PEM format. The OpenSSL PEM format is not formally defined and should only be used by applications which use the OpenSSL libraries to implement security. Other private key formats may exist.

The base name of the Private Key file shall be the same as the base file name for the matching Certificate file stored in the ./certs directory.

A recommended naming convention is:

<CommonName> [<Thumbprint>].(der | pem | pfx)

where the CommonName is the CommonName of the Certificate and the Thumbprint is the SHA1 hash of the certificate formatted as a hexadecimal string.

F.2 Installation directories on Windows

The *LocalDiscoveryServer* executable shall be installed in the following location:

```
%CommonProgamFiles%\OPC Foundation\UA\Discovery
```

where %CommonProgamFiles% is the value of the *CommonProgamFiles* environment variable on 32-bit systems. On 64-bit systems, the value of the *CommonProgamFiles(x86)* environment variable is used instead.

The configuration files used by the *LocalDiscoveryServer* executable shall be installed in the following location:

```
%CommonApplicationData%\OPC Foundation\UA\Discovery
```

where %CommonApplicationData% is the location of the application data folder shared by all users. The exact location depends on the operating system, however, under Windows 7 or later the common application data folder is usually C:\ProgramData.

The certificates stores used by the *LocalDiscoveryServer* shall be organized as described in Clause F.1. The root of the certificates stores shall be in the following location:

```
%CommonApplicationData%\OPC Foundation\UA\pki
```

Annex G
(normative)

Application installation process

G.1 Provisioning with Pull Management

Applications that use Pull Management (see 7.2) to initialize their configuration need to know the location of the *CertificateManager* which they can use to request *Certificates* and download Trust Lists. This location may be auto-discovered via mDNS by looking for servers with the GDS capability (see Annex D) or by providing a URL via and out-of-band mechanism such as e-mail or a web page.

Once the location is known, the *Application* can connect to the *CertificateManager* and establish a secure channel. This will require that the Application trust the *Certificate* provided by the *CertificateManager* even if it is not in the Application's *Trust List*. If there is an interactive user, the Application should warn the user before proceeding. If there is no interactive user, the Application should ensure the domain in the URL matches one of the domains in the *Certificate*.

In some cases, the Application distributor or installer will know the CA used to sign the *Certificate* used by the *CertificateManager* and can add this CA to the Application's *Trust List* during installation. If practical, this approach provides the best protection against accidental registration with rogue *CertificateManagers.*

After establishing a secure channel with the *CertificateManager*, the Application shall provide user credentials which allow it to register new applications and request new *Certificates.* The credentials may be provided by prompting a user or they may be one time use credentials delivered via some out of band mechanism such as a web site during the installation process.

For embedded systems, it can be impractical to enter user credentials. As an alternative, a unique *ApplicationInstance Certificate* can be provided during manufacture and the *Certificate* or a unique identifier for the *Certificate* should be provided to the device installer. The installer would then register the unique identifier or *Certificate* with the *CertificateManager* which would allow the device to request a new *Certificate* by creating a Secure Channel with the manufacturer's *Certificate.*

Once an Application has received its first *Certificate* then the *Certificate* can be used in lieu of user credentials when the Application needs to renew its *Certificate* or update its Trust List.

G.2 Provisioning with Push Management

Servers that use Push Management (see 7.3) to initialize their configuration shall have a default *Certificate* assigned before the Push Management process can start.

In addition, Servers shall go into a "provisioning state" that makes it possible for remote clients to update the security configuration via the *ServerConfiguration Object* (see 7.7.2). When a *Server* is in the "provisioning state" it should limit the available functionality.

Once a Server has been configured, it automatically leaves the "provisioning state". This step is necessary to ensure that security is not compromised.

A possible workflow for implementing the "provisioning state" includes:

1) A flag in the configuration file that defaults to ON;
2) Always allow Clients to connect securely if the *Trust List* is empty;

3) Connect to the Server and provide administrator credentials where:
   – Toggle a physical switch on the device which enables access for a short period or
   – Provide one-time use password specified via an out-of-band mechanism;
4) Provide a new Certificate (optional) and *Trust List*;
5) Set the configuration flag to OFF.

Subsequent updates to Trust Lists or *Certificates* can be allowed if the Client has a trusted *Certificate* and valid administrator credentials.

In some cases, the *Application* distributor or installer will know the CA used to sign the *Certificate* used by the *CertificateManager* and can add this CA to the Application's *Trust List* during installation. If practical, this approach provides the best protection against accidental configuration by malicious Clients.

If the device is automatically discovered by the *CertificateManager*, the *CertificateManager* needs some way to ensure that the device belongs on the network. The manufacturer can provide a unique *ApplicationInstance Certificate* during manufacture and provide the serial numbers to the device installer. The installer would then register the serial number or *Certificate* with the *CertificateManager*. When the *CertificateManager* discovers the device, it would check that the *Certificate* is for one of the pre-authorized devices and continue with automatic provisioning of the device.

G.3 Setting permissions

If a Private Key is stored on a regular file system, it shall be protected from unauthorized access. This is best done by setting operating system permissions on the private key file that deny read/write access to anyone who is not using an account authorized to run the *Application*.

In some cases, additional protection can be added by protecting the Private Key with a password. Saving Private Key passwords in files should be avoided. This mode may also work in conjunction with "smart cards" that use hardware to protect the Private Key.

In addition to the Private Key, *Applications* shall be protected from unauthorized updates to their *Trust List*. This can also be done by setting operating system permissions on the directory where the Trust List is stored that deny write access to anyone who is not using an account authorized to administer the *Application*.

Finally, *Applications* may depend on one or more configuration files and/or databases which tell them where there *Trust List* and Private Key can be found. The source of any security-related configuration information shall be protected from unauthorized updates. The exact mechanism used to implement these protections depends on the source of the information.

Annex H
(informative)

Comparison with RFC 7030

H.1 Overview

RFC 7030 (Enrolment over Secure Transport or EST) defines a mechanism for the distribution of *Certificates* to devices. This appendix summarizes the capabilities provided by EST and how the same capabilities are provided by the *CertificateManager* defined in Clause 7.

H.2 Obtaining CA Certificates

In EST, a web operation returns the CA certificates. In OPC UA, the CA *Certificates* are returned when the *CertificateManager* client reads the *Trust List* assigned to the application from the *CertificateManager*. Prior to these operations the *Client* should verify that the server is authorized to provide CAs. Table H.1 compares how EST clients verify the EST server with how *CertificateManager* clients verify a *CertificateManager*.

Table H.1 – Verifying that a Server is allowed to provide Certificates

EST	OPC UA
Compare the URL for the EST server with the HTTPS certificate returned in the TLS handshake.	Compare the URL for the *CertificateManager* with the OPC UA *Certificate* returned in *GetEndpoints*.
Preconfigure the client to trust the EST *Server's* HTTPS certificate.	Preconfigure the client by adding the *CertificateManager Certificate* to the client *Trust List*.
Manual approval of the CA *Certificate* after comparing the certificate with out of band information.	Manual approval of the *CertificateManager Certificate* after comparing the *Certificate* with out of band information.
Pre-shared credentials for use with certificate-less TLS.	No equivalent.

H.3 Initial enrolment

In EST, a web operation is used to enrol a client. The EST server authenticates and authorizes the EST client before allowing the operation to proceed. In OPC UA, a *Method* is used to request a *Certificate*. The *CertificateManager* also authenticates and authorizes the client before allowing the operation to proceed. Table H.2 compares how EST servers verify the EST client with how a *CertificateManager* verifies a *CertificateManager* client.

Table H.2 – Verifying that a Client is allowed to request Certificates

EST	OPC UA
TLS with a client certificate which is previously issued by the EST server.	The *CertificateManager* client has a previously certificate previously issued by the GDS.
TLS with a previously installed certificate which is trusted by the EST server.	The *CertificateManager* client has a certificate which is trusted by the *CertificateManager*.
Shared credentials distributed out of band which are used for certificate-less TLS.	No equivalent.
HTTPS username/password authentication.	The *CertificateManager* client provides appropriate user credentials when it establishes the session.

H.4 Client Certificate reissuance

In EST, a certificate issued by the EST server can be used as an HTTPS client certificate. This can be used to authorize the re-issue of the certificate. In OPC UA, a certificate issued by the GDS can be used to establish a secure channel. This would then allow the GDS client to request that the certificate be re-issued.

In both EST and OPC UA clients can fall back to the authentication mechanisms used for Initial Enrolment if it is not possible to use the current certificate to establish a secure channel with the server.

H.5 Server key generation

Both EST and OPC UA allow clients to request new private keys. Both specifications require that encryption be used when returning private key data.

EST allows clients to explicitly request that separate encryption be applied to the private key. The algorithms are specified in the CSR (certificate signing request).

OPC UA allows clients to password protect the key (which uses encryption), but it does not allow the client to directly specify the algorithm used. That said, the envelope used to transport private keys can be specified with the *PrivateKeyFormat* parameter and the set of envelope formats supported by the *CertificateManager* is published in the *AddressSpace*. It is expected that the envelope format will specify the algorithms used either by explicitly encoding the algorithm within the envelope or as part of the definition of the envelope.

H.6 Certificate Signing Request (CSR) attributes request

EST allows the client to request the list of CSR attributes the EST server supports. The attributes can indicate what additional metadata the client can provide or the algorithms that will be used.

In OPC UA, the CSR metadata required is fixed by the specification and there is no mechanism to publish extensions. Clients are free to include additional metadata in the CSR; however, the *CertificateManager* may ignore it.

There is no mechanism in OPC UA to publish the algorithms which need to be used for the CSR; however, the *CertificateManager* will reject CSRs that do not meet its requirements.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 12: Services globaux et de découverte

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-12 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/711/FDIS	65E/723/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette Norme internationale.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2.

Dans l'ensemble du présent document et dans les autres parties de la série IEC 62541, certaines conventions de document sont utilisées:

Le format *italique* est utilisé pour mettre en évidence un terme défini ou une définition qui apparaît à l'article "Termes et définitions" dans l'une des parties de la série IEC 62541.

Le format *italique* est également utilisé pour mettre en évidence le nom d'un paramètre d'entrée ou de sortie de service, ou le nom d'une structure ou d'un élément de structure habituellement défini dans les tableaux.

Par ailleurs, les *termes* et les *noms en italique* sont, à quelques exceptions près, écrits en camel-case (pratique qui consiste à joindre, sans espace, les éléments des mots ou expressions composés, la première lettre de chaque élément étant en majuscule). Par exemple, le terme défini est *AddressSpace* et non Espace d'adressage. Cela permet de mieux comprendre qu'il existe une définition unique pour *AddressSpace*, et non deux définitions distinctes pour Espace et pour Adressage.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Architecture unifiée OPC*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives au document recherché. A cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

ARCHITECTURE UNIFIÉE OPC –

Partie 12: Services globaux et de découverte

1 Domaine d'application

La présente partie de l'IEC 62541 spécifie la manière dont les *Clients* et les *Serveurs* de l'Architecture Unifiée OPC (OPC UA) interagissent avec les *DiscoveryServers* lorsqu'ils sont utilisés dans différents scénarios. Elle définit les exigences pour le *LocalDiscoveryServer*, le *LocalDiscoveryServer-ME* et le *GlobalDiscoveryServer*. Elle définit également les modèles d'information pour la gestion des *Certificats*, la gestion des *KeyCredentials* et les *Services d'Autorisation*.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and concepts* (disponible en anglais seulement)

IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model* (disponible en anglais seulement)

IEC 62541-3, *Architecture unifiée OPC – Partie 3: Modèle d'espace d'adressage*

IEC 62541-4, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'Information*

IEC 62541-6, *Architecture unifiée OPC – Partie 6: Mappings*

IEC 62541-7, *Architecture unifiée OPC – Partie 7: Profils*

IEC 62541-9, *Architecture unifiée OPC – Partie 9: Alarmes et Conditions*

IEC 62541-14, *Architecture unifiée OPC – Partie 14: PubSub*

X.500: ISO/IEC 9594-1:2017, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – L'annuaire – Partie 1: Aperçu général des concepts, modèles et services*

IETF RFC 1035, *DNS-Name: Domain Names – Implementation and Specification*
http://www.ietf.org/rfc/rfc1035.txt

IETF RFC 2986, *PKCS #10: Certification Request Syntax Specification*
http://www.ietf.org/rfc/rfc2986.txt

IETF RFC 3927, *Auto-IP: Dynamic Configuration of IPv4 Link-Local Addresses*
http://www.ietf.org/rfc/rfc3927.txt

IETF RFC 5958, *Asymmetric Key Packages*
http://www.ietf.org/rfc/rfc5958.txt

IETF RFC 6762, *mDNS: Multicast DNS*
http://www.ietf.org/rfc/rfc6762.txt

IETF RFC 6763, *DNS-SD: DNS Based Service Discovery*
http://www.ietf.org/rfc/rfc6763.txt

IETF RFC 7030, *Enrollment over Secure Transport*
http://www.ietf.org/rfc/rfc7030.txt

PKCS #12: *Personal Information Exchange Syntax*
http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11301-wp-pkcs-12v1-1-personal-information-exchange-syntax.pdf

DI: *OPC Unified Architecture for Devices (DI)*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-devices-di/

ADI: *OPC Unified Architecture for Analyzer Devices (ADI)*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-analyzer-devices-adi/

PLCopen: *OPC Unified Architecture / PLCopen Information Model*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-plcopen-information-model/

FDI: *OPC Unified Architecture for FDI*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/opc-unified-architecture-for-fdi/

ISA-95: *ISA-95 Common Object Model*
https://opcfoundation.org/developer-tools/specifications-unified-architecture/isa-95-common-object-model/

3 Termes, définitions, termes abrégés et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1, l'IEC TR 62541-2, l'IEC 62541-3, l'IEC 62541-4, l'IEC 62541-6 et l'IEC 62541-9 ainsi que les suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1.1
groupe de certificats
contexte utilisé pour décrire la *Liste de Confiance* et le(s) *Certificat(s)* associé(s) à une *Application*

3.1.2
demande de certificat
structure à codage *PKCS #10* utilisée pour demander un nouveau *Certificat* auprès d'une *Autorité de Certification*

3.1.3
KeyCredential
identificateur unique et secret utilisés pour accéder à un *Serveur*, un *Service d'Autorisation* ou un *Courtier*

Note 1 à l'article: Un nom d'utilisateur et un mot de passe constituent un exemple d'authentifiant.

3.1.4
KeyCredentialService
application logicielle qui fournit les *KeyCredentials* nécessaires pour accéder à un *Serveur*, un *Service d'Autorisation* ou un *Courtier*

3.1.5
DirectoryService
application logicielle, ou ensemble d'applications, qui stocke et organise les informations sur les ressources telles que les ordinateurs ou les services

3.1.6
DiscoveryServer
Application qui gère la liste des *Serveurs* OPC UA disponibles sur le réseau et qui fournit les mécanismes permettant aux Clients d'obtenir cette liste

3.1.7
DiscoveryUrl
URL d'un *Point d'extrémité* réseau qui fournit les informations exigées pour se connecter à un *Client* ou un *Serveur*

3.1.8
GlobalDiscoveryServer
GDS
DiscoveryServer qui gère la liste des *Applications* OPC UA disponibles dans un domaine administratif

Note 1 à l'article: Un GDS peut également fournir des services de gestion de certificats.

3.1.9
IPAddress
numéro unique attribué à une interface réseau permettant d'acheminer les demandes IP (Internet Protocol) vers cette interface

Note 1 à l'article: L'*IPAddress* d'un hôte peut changer au fil du temps.

3.1.10
LocalDiscoveryServer
LDS
DiscoveryServer qui gère la liste de tous les *Serveurs* qui y sont enregistrés

Note 1 à l'article: En règle générale, les *Serveurs* s'enregistrent auprès du LDS sur le même hôte.

3.1.11
LocalDiscoveryServer-ME
LDS-ME
LocalDiscoveryServer qui inclut la MulticastExtension

3.1.12
MulticastExtension
extension d'un *LocalDiscoveryServer* qui ajoute la prise en charge du protocole *mDNS*

3.1.13
MulticastSubnet
réseau permettant l'envoi de paquets de multidiffusion vers tous les nœuds connectés au réseau

Note 1 à l'article: Le *MulticastSubnet* n'est pas nécessairement identique à un sous-réseau TCP/IP.

3.1.14
service de réseau
ressource sécurisée sur un réseau qui fournit une fonctionnalité utilisée par les *Clients* et/ou les *Serveurs*

Note 1 à l'article: Un *Service d'Autorisation* (AS) est un exemple de *service de réseau*.

3.1.15
ServerCapabilityIdentifier
court identificateur qui identifie de manière unique un ensemble de fonctionnalités décelables prises en charge par un *Serveur*

Note 1 à l'article: La liste des ServerCapabilityIdentifiers actuellement définis est fournie à l'Annexe D.

3.2 Termes abrégés et symboles

API	application programming interface (Interface de programmation d'application)
CA	certificate authority (Autorité de certification)
CRL	certificate revocation list (Liste de révocation de certificat)
CSR	certificate signing request (Demande de signature de certificat)
DER	distinguished encoding rules (Règles de codage distinctives)
DNS	Domain Name System (Système de noms de domaine)
EST	Enrolment over Secure Transport (Protocole EST)
GDS	Global Discovery Server (Serveur de découverte global)
IANA	Internet Assigned Numbers Authority (Autorité IANA)
LDAP	Lightweight Directory Access Protocol (Protocole LDAP)
LDS	Local Discovery Server (Serveur de découverte local)
LDS-ME	Local Discovery Server with the Multicast Extension (Serveur de découverte local avec extension de multidiffusion)
mDNS	Multicast Domain Name System (Système de noms de domaine de multidiffusion)
NAT	network address translation (Traduction d'adresse réseau)
PEM	Privacy Enhanced Mail (Courrier à confidentialité améliorée)
PFX	Personal Information Exchange (Echange d'informations personnelles)
PKCS	Public Key Cryptography Standards (Normes de cryptographie à clé publique)
SHA1	Secure Hash Algorithm (Algorithme de hachage sécurisé)
SSL	Secure Sockets Layer (Protocole SSL)
TLS	Transport Layer Security (Protocole TLS)
UA	Unified Architecture (Architecture unifiée)
UDDI	Universal Description, Discovery and Integration (Norme UDDI)

3.3 Conventions pour les espaces de noms

Le présent document utilise plusieurs espaces de noms pour définir les *Nœuds*. Les termes abrégés suivants sont utilisés dans les définitions de ces *Nœuds*.

CORE http://opcfoundation.org/UA/

GDS http://opcfoundation.org/UA/GDS/

L'espace de nom par défaut de chaque *Nœud* est défini en haut du tableau. L'ensemble des *BrowseNames* du tableau utilise l'espace de nom par défaut, sauf lorsque le *BrowseName* est précédé de l'une des abréviations ci-dessus.

L'*Objet NamespaceMetadataType* de l'espace de nom GDS est défini dans le Tableau 1.

Tableau 1 – Définition de l'Objet NamespaceMetadataType GDS

Attribut	Valeur		
BrowseName	http://opcfoundation.org/UA/GDS/		
Namespace	GDS (voir 3.3)		
TypeDefinition	NamespaceMetadataType défini dans l'IEC 62541-5.		
Références	**NodeClass**	**BrowseName**	**Valeur**
HasProperty	Variable	NamespaceUri	http://opcfoundation.org/UA/GDS/
HasProperty	Variable	NamespaceVersion	1.04
HasProperty	Variable	NamespacePublicationDate	2016-12-31

4 Processus de découverte

4.1 Vue d'ensemble

Le processus de découverte permet à une application de trouver d'autres applications sur le réseau avant de découvrir comment s'y connecter. Noter que cette explication repose sur les concepts de découverte définis dans l'IEC 62541-4. Les applications décelables sont généralement des *Serveurs*; toutefois, certains *Clients* prennent en charge les connexions inverses décrites dans l'IEC 62541-6 et souhaitent que les *Serveurs* soient capables de les découvrir.

Les *Clients* et les *Serveurs* peuvent se trouver sur le même hôte ou sur différents hôtes du même sous-réseau, voire sur des emplacements radicalement distincts dans un domaine administratif. Les paragraphes suivants décrivent les différentes configurations ainsi que la manière dont la découverte peut s'effectuer.

Les mécanismes permettant aux *Clients* de découvrir les *Serveurs* sont spécifiés en 4.3.

Les mécanismes permettant aux *Serveurs* de devenir décelables sont spécifiés en 4.2.

Les *Services* de *Découverte* sont définis dans l'IEC 62541-4. Ils sont mis en œuvre par des *Serveurs* individuels et des *DiscoveryServers* dédiés. Les *DiscoveryServers* dédiés suivants permettent aux applications de découvrir les applications OPC UA enregistrées dans différentes situations:

- un *LocalDiscoveryServer* (LDS) gère les informations de découverte pour toutes les applications qui y sont enregistrées, en général toutes les applications disponibles sur l'hôte sur lequel il s'exécute;

- un *LocalDiscoveryServer* avec la *MulticastExtension* (LDS-ME) gère les informations de découverte pour toutes les applications qui ont été annoncées sur le *MulticastSubnet* local;
- un *GlobalDiscoveryServer* (GDS) gère les informations de découverte pour les applications disponibles dans un domaine administratif.

Les LDS et les LDS-ME sont spécifiés à l'Article 5. Les GDS sont spécifiés à l'Article 6.

L'Annexe A fournit des recommandations quant à l'utilisation de pare-feu. L'Annexe B définit les nœuds de découverte globale. L'Annexe F définit un répertoire pour les applications qui stockent leurs *Certificats* sur un système de fichiers. L'Annexe G fournit des instructions pour la procédure d'installation de l'application. L'Annexe H compare la gestion de Certificats décrite dans le présent document à la RFC 7030.

4.2 Enregistrement et annonce d'Applications

4.2.1 Vue d'ensemble

Le présent paragraphe décrit la manière dont une application s'enregistre de sorte à pouvoir être découverte. La plupart des *Applications* souhaitent être découvertes par les autres applications. Il convient que les *Applications* qui ne souhaitent pas être ouvertement découvertes ne s'enregistrent pas auprès d'un *DiscoveryServer*. Dans ce cas, il convient que ces *Applications* publient uniquement une *DiscoveryUrl* à l'aide de certains mécanismes hors bande pour être découvertes par des *Applications* spécifiques.

4.2.2 Hôtes avec un LocalDiscoveryServer

Les applications s'enregistrent auprès du LDS sur le même hôte si elles souhaitent être découvertes. L'enregistrement garantit que les applications sont visibles pour une découverte locale (voir 4.3.4) et pour une découverte de *MulticastSubnet* si le LDS est un LDS-ME (voir 4.3.5).

La norme OPC UA (IEC 62541-4) définit un *Service RegisterServer2* qui fournit des informations d'enregistrement supplémentaires. Toutes les *Applications* ainsi que le *LocalDiscoveryServer* doivent prendre en charge le *Service RegisterServer2* et, à des fins de rétrocompatibilité, l'ancien *Service RegisterServer*. Si une *Application* rencontre un ancien LDS qui renvoie une erreur *Bad_ServiceUnsupported* lors de l'appel du *Service RegisterServer2*, elle doit réessayer avec le *Service RegisterServer*.

Le *Service RegisterServer2* permet à l'*Application* de spécifier zéro ou plusieurs *ServerCapabilityIdentifiers*. Les *ServerCapabilityIdentifiers* sont des identificateurs de chaîne courts pour les fonctionnalités OPC UA notoirement connues. Les *Applications* peuvent utiliser ces identificateurs comme filtre pendant la découverte.

L'ensemble des *ServerCapabilityIdentifiers* connus est spécifié à l'Annexe D et se limite aux fonctionnalités jugées suffisamment importantes pour être rapportées avant qu'une application n'établisse une connexion. Par exemple, la prise en charge du modèle d'information GDS ou du modèle d'information d'Alarmes constitue une fonctionnalité de *Serveur* disposant d'un *ServerCapabilityIdentifier* défini.

Avant qu'elle ne s'enregistre auprès du LDS, il convient que l'application appelle le *Service GetEndpoints* et choisisse le point d'extrémité le plus sécurisé pris en charge par le LDS avant d'appeler le *Service RegisterServer2* ou *RegisterServer*.

L'enregistrement auprès d'un LDS ou d'un LDS-ME est représenté à la Figure 1.

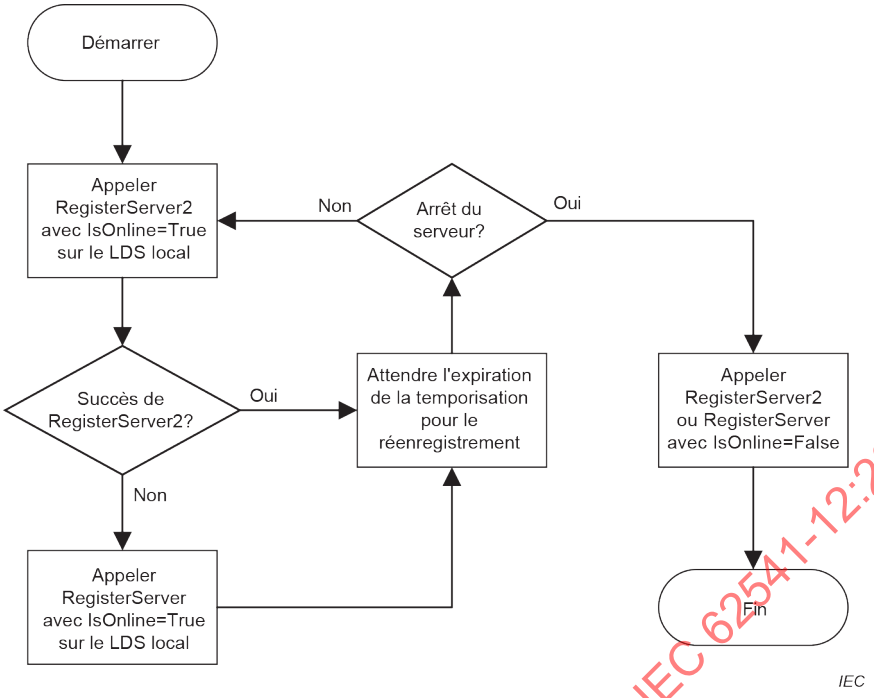


Figure 1 – Processus d'enregistrement auprès d'un LDS

Pour plus d'informations sur le temporisateur de réenregistrement et le fanion *IsOnline*, voir l'IEC 62541-4.

4.2.3 Hôtes sans LocalDiscoveryServer

Les systèmes dédiés (en général intégrés) avec précisément un *Serveur* installé peuvent ne pas avoir de LDS séparé. Ces *Serveurs* doivent devenir leur propre LDS ou LDS-ME en mettant en œuvre les *Services FindServers* et *GetEndpoints* à l'adresse notoire d'un LDS. Par ailleurs, il convient qu'ils s'annoncent sur le *MulticastSubnet* au moyen d'une *MulticastExtension* de base. Cela exige un sous-réseau de Répondeur mDNS de petite taille (voir *mDNS* et Annexe C) qui annonce le *Serveur* et répond aux sondes mDNS. Le *Serveur* peut ne pas fournir la mise en cache et la résolution d'adresse mises en œuvre par un Répondeur mDNS intégral.

4.3 Processus de découverte permettant aux Clients de trouver des Serveurs

4.3.1 Vue d'ensemble

Le processus de découverte permet aux *Clients* de trouver des *Serveurs* sur le réseau avant de découvrir comment s'y connecter. Lorsque le *Client* dispose de ces informations, il peut les sauvegarder et les utiliser pour se reconnecter directement au *Serveur* sans passer par le processus de découverte. Il convient que les *Clients* qui ne peuvent pas se connecter avec les informations de connexion sauvegardées prennent pour hypothèse que la configuration du *Serveur* a changé et qu'ils répètent donc le processus de découverte.

Un *Client* dispose de plusieurs choix pour trouver des *Serveurs*:

- la découverte hors bande (c'est-à-dire l'entrée dans une GUI) de la *DiscoveryUrl* d'un *Serveur*;
- l'appel du *Service FindServers* sur le LDS installé sur l'hôte du *Client*;
- l'appel du *service FindServers* sur un LDS distant sur lequel le *HostName* de l'hôte distant a été saisi manuellement;
- l'appel du *service FindServersOnNetwork* (voir l'IEC 62541-4) sur le LDS-ME installé sur l'hôte du *Client*;

- la prise en charge de la fonctionnalité LDS-ME en local sur le *Client*;
- la recherche de *Serveurs* connus d'un *GlobalDiscoveryServer*.

La *DiscoveryUrl* fournit toutes les informations nécessaires au *Client* pour se connecter à un *DiscoveryEndpoint* (voir 4.3.3).

4.3.2 Sécurité

Il convient que les *Clients* connaissent les *DiscoveryServers* malveillants susceptibles de les rediriger vers des *Serveurs* malveillants. Les *Clients* peuvent utiliser le certificat de serveur SSL/TLS (si disponible) pour vérifier que le *DiscoveryServer* est un serveur de confiance et/ou s'assurer qu'ils font confiance à tous les *Serveurs* fournis par le *DiscoveryServer*. Voir l'IEC TR 62541-2 pour obtenir une explication détaillée de ces sujets.

4.3.3 Découverte simple avec une DiscoveryUrl

Chaque *Serveur* dispose d'une ou de plusieurs *DiscoveryUrls* qui permettent d'accéder à ses *Points d'extrémité*. Lorsque le *Client* obtient (au moyen d'une saisie manuelle dans un formulaire, par exemple) la *DiscoveryUrl* du *Serveur*, il lit les *EndpointDescriptions* à l'aide du *Service GetEndpoints* défini dans l'IEC 62541-4.

Le processus de découverte pour ce scénario est représenté à la Figure 2.

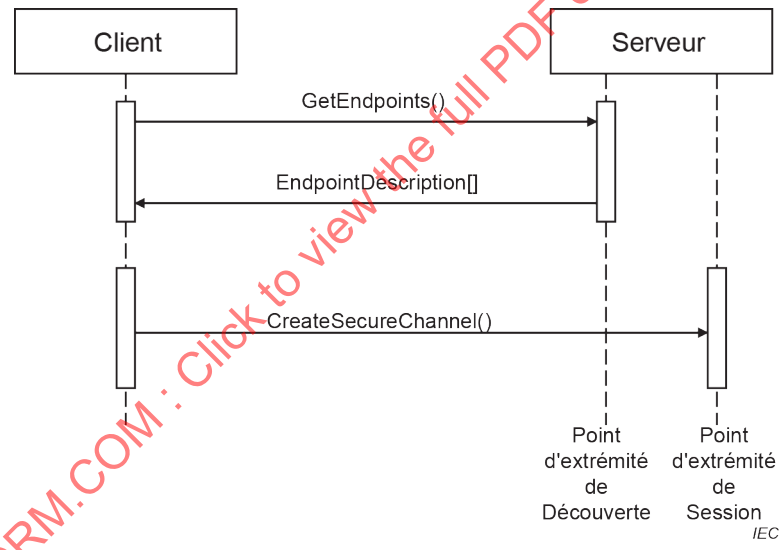


Figure 2 – Processus de découverte simple

4.3.4 Découverte locale

Dans de nombreux cas, les *Clients* ne savent pas quels *Serveurs* existent, mais savent éventuellement quels hôtes sont susceptibles d'héberger des *Serveurs*. Dans ce cas, le *Client* recherche le *LocalDiscoveryServer* sur un hôte en construisant une *DiscoveryUrl* à l'aide des adresses notoires définies dans l'IEC 62541-6.

Si un *Client* trouve un *LocalDiscoveryServer*, il appelle le *Service FindServers* sur le LDS pour obtenir une liste des *Serveurs* et de leurs *DiscoveryUrls*. Le *Client* appelle alors le *Service GetEndpoints* pour l'un des *Serveurs* renvoyés. Le processus de découverte pour ce scénario est représenté à la Figure 3.

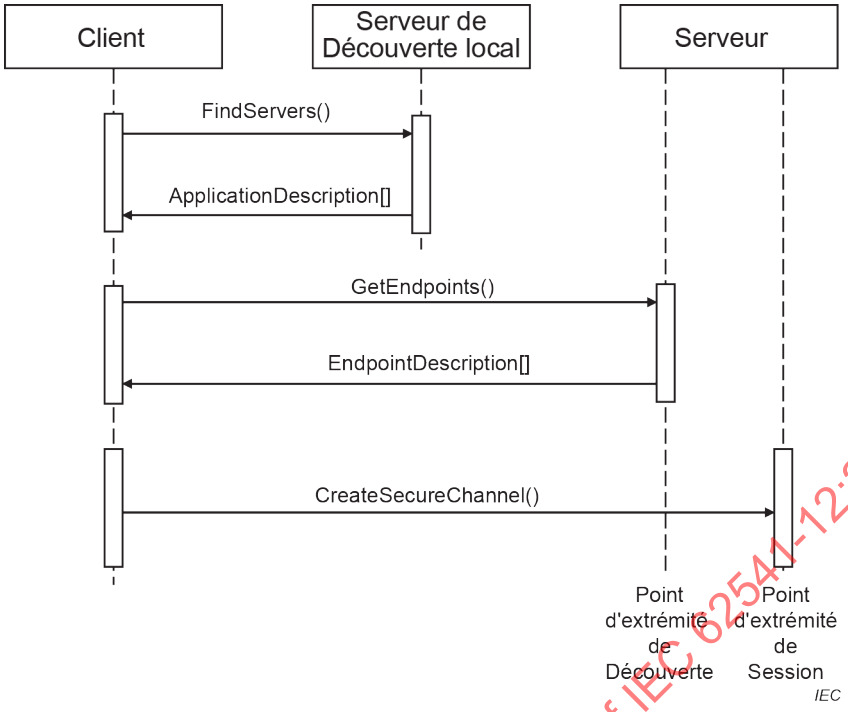


Figure 3 – Processus de découverte locale

4.3.5 Découverte de MulticastSubnet

Dans certains cas, les *Clients* ne savent pas quels hôtes hébergent des *Serveurs*. Dans ce cas, le *Client* recherche un *LocalDiscoveryServer* avec la *MulticastExtension* sur son hôte local et demande une liste de *DiscoveryUrls* pour les *Serveurs* et les *DiscoveryServers* disponibles sur le *MulticastSubnet*.

Le processus de découverte pour ce scénario est représenté à la Figure 4.

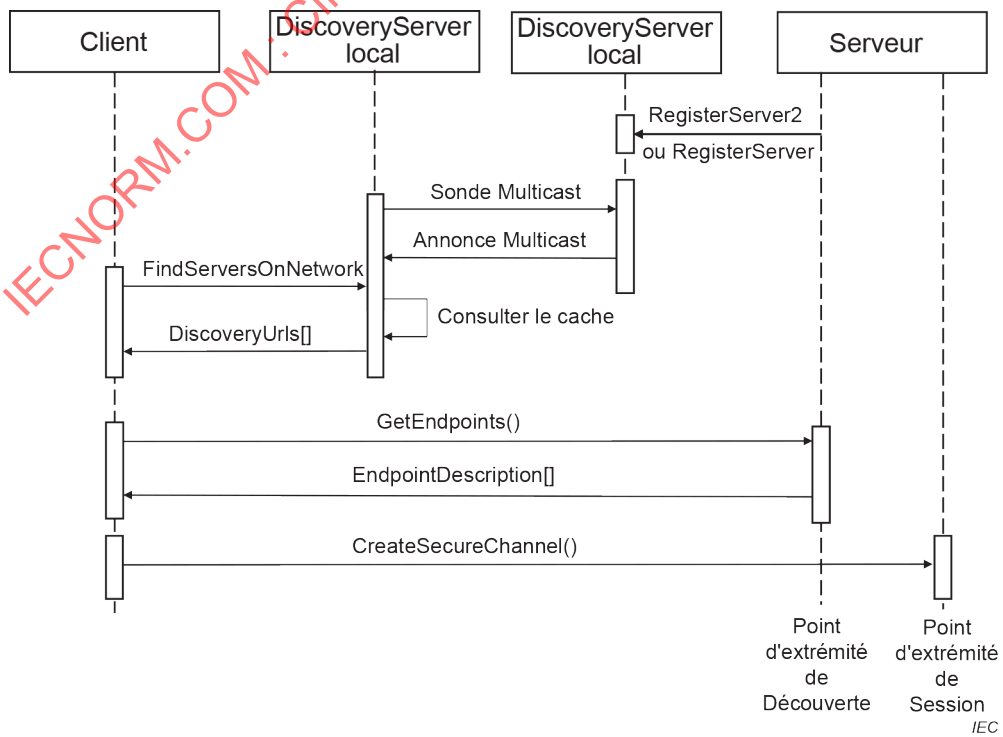


Figure 4 – Processus de découverte de MulticastSubnet

Dans ce scénario, le *Serveur* utilise le *Service RegisterServer2* pour indiquer à un *LocalDiscoveryServer* d'annoncer le *Serveur* sur le *MulticastSubnet*. Le *Client* reçoit la *DiscoveryUrl* et les *ServerCapabilityIdentifiers* pour le *Serveur* lorsqu'il appelle le *Service FindServersOnNetwork* avant de se connecter directement au *Serveur*. Lorsqu'un *Client* appelle le *Service FindServers*, il reçoit uniquement les *Serveurs* qui s'exécutent sur le même hôte que celui du LDS.

Les *Clients* qui s'exécutent sur un système intégré peuvent ne pas avoir de LDS-ME disponible sur le système. Ces *Clients* peuvent prendre en charge un Répondeur mDNS qui identifie la manière dont les concepts OPC UA sont associés par mapping avec les messages mDNS et gère le même tableau de *Serveurs* que celui géré par le LDS-ME. Ce mapping est décrit à l'Annexe C.

4.3.6 Découverte globale

Un GDS est un *Serveur* OPC UA qui permet aux *Clients* de rechercher des *Serveurs* dans un domaine administratif. Il peut également proposer des Services de Certificat (voir Article 7). Il fournit des *Méthodes* permettant aux applications de rechercher d'autres applications (voir Article 6). Pour accéder au GDS, le *Client* crée une *Session* avec le *GDS* et utilise le service d'*Appel* pour invoquer la *Méthode QueryApplications* (voir 6.3.11). La *Méthode QueryServers* est similaire au *Service FindServers*, sauf qu'elle propose des critères de recherche et de filtrage plus avancés. Le processus de découverte est représenté à la Figure 5.

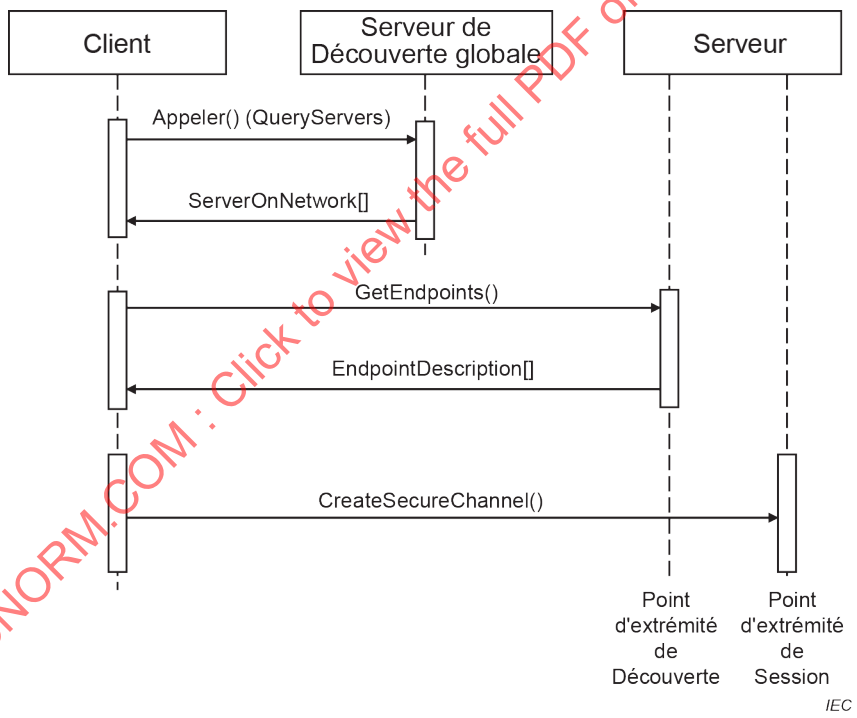


Figure 5 – Processus de découverte globale

Le GDS peut être couplé avec l'une des architectures réseau précédentes. Pour chaque *MulticastSubnet*, un ou plusieurs LDS peuvent être enregistrés auprès d'un GDS.

Le *Client* peut également être configuré avec l'URL du GDS à l'aide d'un mécanisme hors bande.

Le processus de découverte complet est représenté à la Figure 6.

4.3.7 Processus de découverte combiné pour les Clients

Les cas d'utilisation décrits dans les paragraphes précédents impliquent que les *Clients* doivent effectuer plusieurs choix lorsqu'il est nécessaire qu'ils se connectent à un *Serveur*. Ces choix sont combinés à la Figure 6.

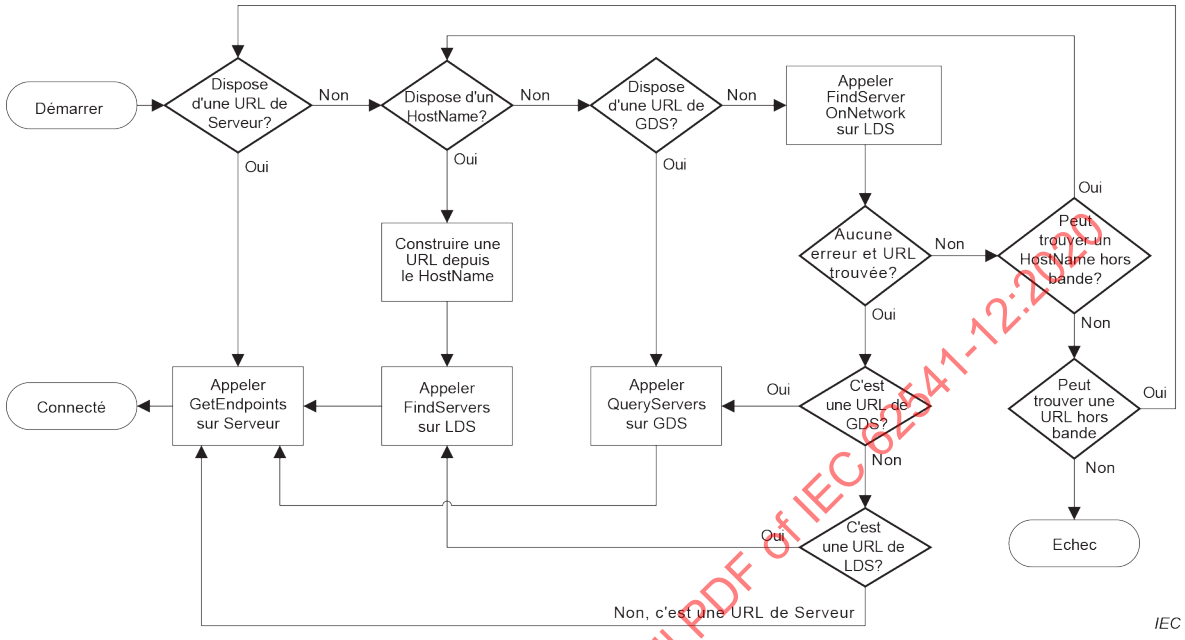


Figure 6 – Processus de découverte pour les Clients

Le *Service FindServersOnNetwork* peut être appelé sur le LDS local, mais il peut également être appelé sur un LDS distant faisant partie d'un *MulticastSubnet* différent.

Un mécanisme hors bande permet de trouver une URL ou un *HostName* non décrit dans la présent document. Par exemple, un utilisateur peut saisir manuellement une URL ou utiliser des API spécifiques au système pour parcourir le voisinage du réseau.

Un *Client* qui exécute le processus de découverte peut sauvegarder l'URL découverte. Si le *Client* redémarre ultérieurement, il peut utiliser cette URL et contourner le processus de découverte. Si la reconnexion échoue, le *Client* doit renouveler le processus.

5 Serveur de découverte local

5.1 Vue d'ensemble

Il convient que chaque hôte sur lequel plusieurs applications décelables sont susceptibles d'être installées dispose d'un *LocalDiscoveryServer* autonome installé. Le *LocalDiscoveryServer* doit exposer un ou plusieurs *Points d'extrémité* qui prennent en charge les services *FindServers* et *GetEndpoints* définis dans l'IEC 62541-4 pour toutes les applications de l'hôte. En outre, le *LocalDiscoveryServer* doit fournir au moins un *Point d'extrémité* qui met en œuvre le *Service RegisterServer* pour ces applications.

Dans les systèmes (en général intégrés) avec précisément un *Serveur* installé, ce *Serveur* peut également être le LDS (voir 4.2.3).

Un LDS-ME annonce toutes les applications qu'il connaît sur le *MulticastSubnet* local. Pour ce faire, un *LocalDiscoveryServer* prend en charge le *Service RegisterServer2* défini dans l'IEC 62541-4. A des fins de rétrocompatibilité, un *LocalDiscoveryServer* prend également en charge le *Service RegisterServer* défini dans l'IEC 62541-4.

Il convient que chaque hôte sur lequel des Applications OPC UA (Clients et Serveurs) sont installées dispose d'un *LocalDiscoveryServer* avec une *MulticastExtension*.

La *MulticastExtension* intègre la fonctionnalité du Répondeur mDNS décrit dans la spécification Multicast DNS (mDNS) (voir *mDNS*). En outre, le *LocalDiscoveryServer* qui prend en charge la *MulticastExtension* prend également en charge le *Service FindServersOnNetwork* décrit dans l'IEC 62541-4.

5.2 Considérations relatives à la sécurité pour Multicast DNS

La spécification Multicast DNS (mDNS) est utilisée pour de nombreuses applications commerciales et grand public. Elle présente l'avantage de faire l'objet de mises en œuvre existantes; les administrateurs système peuvent toutefois choisir de désactiver les opérations Multicast DNS. Par conséquent, les *Applications* ne doivent pas reposer sur des fonctionnalités Multicast DNS.

Du fait de leur nature, les opérations Multicast DNS ne sont pas sécurisées; il convient donc de les désactiver dans les environnements où un attaquant pourrait provoquer des problèmes en se substituant à un autre hôte. L'activation de la sécurité OPC UA et l'utilisation de *TrustLists* de *Certificats* par toutes les *Applications* pour contrôler les accès permettent de réduire ce risque le plus possible.

6 Serveur de découverte global

6.1 Vue d'ensemble

Le *LocalDiscoveryServer* est utile pour les réseaux dont les noms d'hôte peuvent être découverts. Toutefois, cela n'est généralement pas le cas des systèmes de grande taille disposant de plusieurs serveurs sur de multiples sous-réseaux. Il est par conséquent nécessaire d'utiliser un *DiscoveryServer* à l'échelle de l'entreprise, appelé *GlobalDiscoveryServer*. Le *GlobalDiscoveryServer* (GDS) est un *Serveur* OPC UA qui permet aux *Clients* de rechercher des *Serveurs* dans le domaine administratif. Il fournit des méthodes permettant aux applications de s'enregistrer et de rechercher d'autres applications.

L'élément essentiel d'un *GlobalDiscoveryServer* (GDS) réside dans le fait qu'il puisse fournir les services de gestion de *Certificats* définis à l'Article 7. Ces services peuvent simplifier la gestion des *Certificats*, y compris dans les systèmes de petite taille ou de taille moyenne; de ce fait, un GDS peut être déployé dans les systèmes de plus petite taille. Différentes mises en œuvre sont prévues. Certaines d'entre elles sont susceptibles de fournir une interface avec un *DirectoryService* existant, tel que LDAP (voir Annexe E). Lorsqu'ils fondent leur normalisation sur une interface OPC UA, les *Clients* OPC UA n'ont pas besoin de connaître les différents *DirectoryServices*.

Si un administrateur enregistre un *LocalDiscoveryServer* auprès du GDS, ce dernier doit régulièrement mettre à jour sa base de données en appelant le service *FindServersOnNetwork* ou *FindServers* sur le LDS. La Figure 7 représente la relation entre un GDS et le LDS-ME ou le LDS.

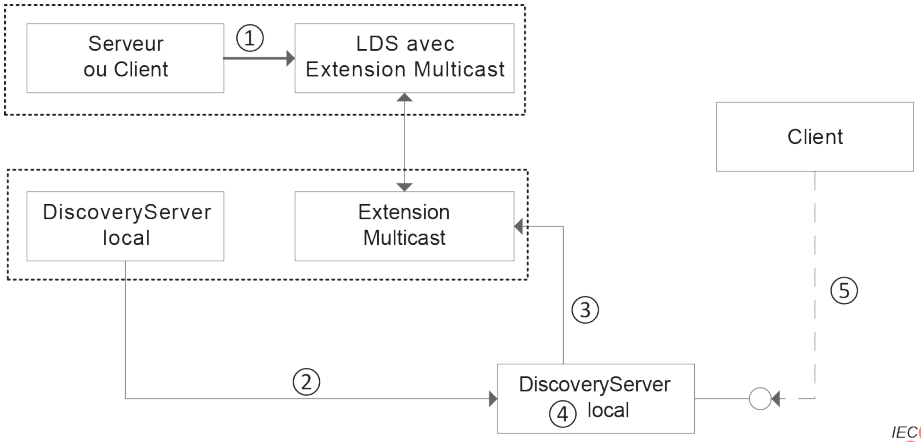


Figure 7 – Relation entre le GDS et les autres composants

Les étapes représentées à la Figure 7 sont les suivantes:

1) Le Serveur appelle le *Service RegisterServer2* sur le LDS qui s'exécute sur la même machine.
2) L'administrateur enregistre les installations du LDS-ME auprès du GDS.
3) Le GDS appelle le *Service FindServersOnNetwork* sur le LDS-ME afin de trouver toutes les applications présentes sur le même *MulticastSubnet*.
4) Le GDS crée un enregistrement pour chaque application renvoyée par le LDS-ME. Ces enregistrements doivent être approuvés avant d'être mis à disposition des *Clients* du GDS. Cette approbation peut être obtenue auprès d'un *Administrateur* ou le GDS peut se connecter au *Serveur* et vérifier qu'il dispose d'un *Certificat* de confiance.
5) Le *Client* appelle la *Méthode QueryServers* sur le GDS pour découvrir les applications.

Le *Modèle d'Information* utilisé pour l'enregistrement et la découverte est représenté en 6.2. Chaque *Client* doit être capable d'appeler la *Méthode QueryServers* pour trouver les applications connues du GDS. Les définitions complètes de chacun des types utilisés sont décrites en 7.5.

6.2 Architectures réseau

6.2.1 Vue d'ensemble

Les mécanismes de découverte définis dans le présent document sont prévus pour être utilisés avec de nombreuses architectures réseau différentes. Les trois architectures suivantes sont représentées:

- *MulticastSubnet* simple;
- *MulticastSubnets* multiples;
- sans *MulticastSubnet* (ou *MulticastSubnets* multiples avec précisément un hôte chacun).

Un *MulticastSubnet* est un segment de réseau où tous les hôtes du segment peuvent recevoir des paquets de multidiffusion des autres hôtes du segment. Un segment de réseau local physique est généralement un *MulticastSubnet*, sauf lorsque l'administrateur a spécifiquement désactivé la communication en multidiffusion. Dans certains cas, plusieurs segments de réseau local physique peuvent être connectés en tant que *MulticastSubnet* simple.

6.2.2 MulticastSubnet simple

L'architecture à *MulticastSubnet* simple est représentée à la Figure 8.

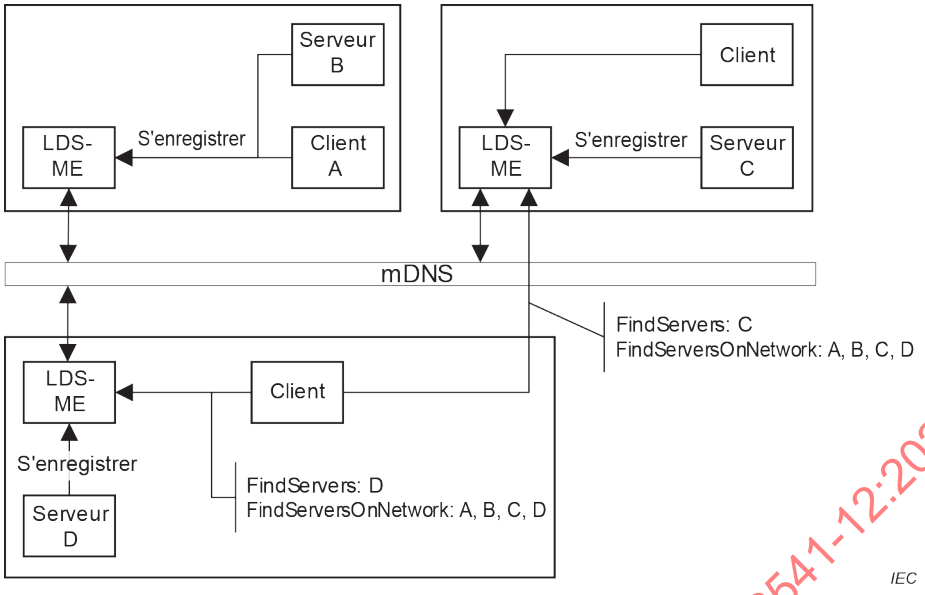


Figure 8 – Architecture à MulticastSubnet simple

Dans cette architecture, chaque hôte possède un LDS-ME et utilise mDNS pour gérer le cache des applications sur le *MulticastSubnet*. Un *Client* peut appeler le *Service FindServersOnNetwork* sur n'importe quel LDS-ME et recevoir le même ensemble d'applications. Lorsqu'un *Client* appelle le *Service FindServers*, il reçoit uniquement les applications qui s'exécutent sur le même hôte que celui du LDS.

6.2.3 MulticastSubnets multiples

L'architecture à *MulticastSubnets* multiples est représentée à la Figure 9.

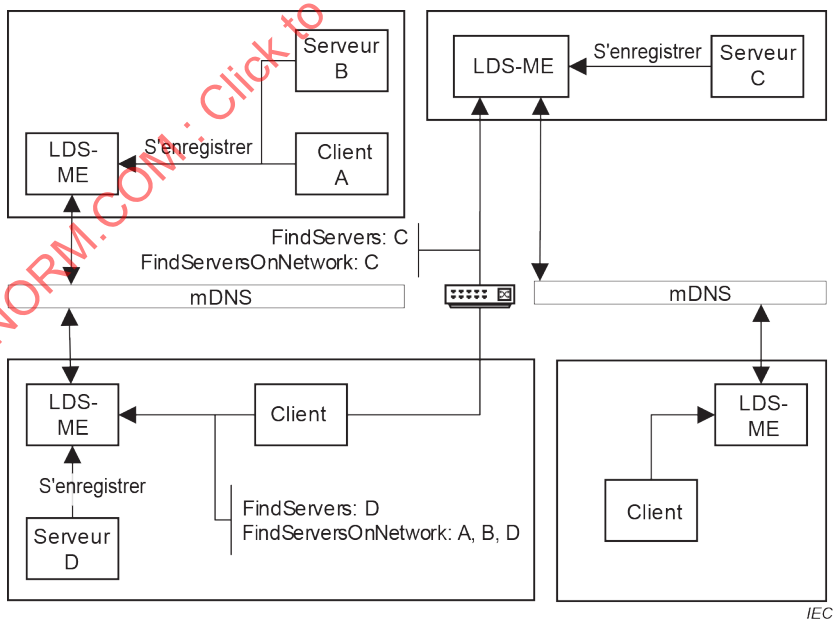


Figure 9 – Architecture à MulticastSubnets multiples

Cette architecture est identique à l'architecture précédente, à la différence que les messages mDNS ne traversent pas les routeurs qui connectent les *MulticastSubnets*. Cela signifie qu'un *Client* appelant le *Service FindServersOnNetwork* reçoit uniquement une liste des applications qui s'exécutent sur les *MulticastSubnets* auxquels le LDS-ME est connecté.

Un *Client* qui souhaite se connecter à un *MulticastSubnet* distant doit utiliser la découverte hors bande (à savoir une saisie manuelle) d'un *HostName* ou d'une *DiscoveryUrl*. Lorsque le *Client* trouve un LDS-ME sur un *MulticastSubnet* distant, il peut utiliser le *Service FindServersOnNetwork* pour découvrir toutes les applications sur ce *MulticastSubnet*.

6.2.4 Sans MulticastSubnet

L'architecture sans *MulticastSubnet* est représentée à la Figure 10.

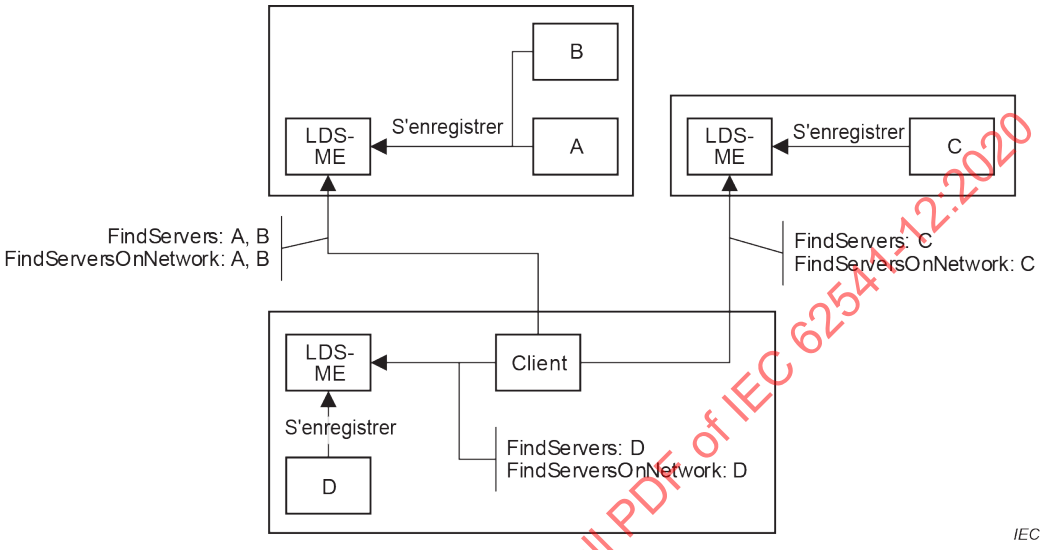


Figure 10 – Architecture sans MulticastSubnet

Dans cette architecture, la spécification mDNS n'est pas du tout utilisée, l'Administrateur ayant désactivé la multidiffusion au niveau du réseau ou ayant mis les fonctionnalités de multidiffusion de chaque LDS-ME à l'arrêt.

Pour découvrir une application, il est nécessaire qu'un *Client* utilise un mécanisme hors bande pour trouver le *HostName* et appeler le *Service FindServers* sur le LDS de cet hôte. Le *Service FindServersOnNetwork* peut également fonctionner; toutefois, il ne renvoie jamais plus de résultats que le *Service FindServers*.

6.2.5 Noms de domaine et MulticastSubnets

La spécification mDNS exige l'annonce d'un nom de domaine pleinement qualifié sur le réseau. Si un *Serveur* n'est pas configuré avec un nom de domaine pleinement qualifié, la spécification mDNS exige l'ajout du domaine "local" de niveau supérieur aux noms de domaine. Le domaine "local" de niveau supérieur indique que le domaine peut uniquement être considéré comme unique au sein du sous-réseau dans lequel le nom de domaine est utilisé. Cela signifie qu'il est nécessaire que le *Client* comprenne que les URL reçues d'un autre LDS-ME que celui présent sur la machine du *Client* peuvent contenir des domaines "locaux" qui sont inaccessibles ou qui se connectent à une machine différente avec le même nom de domaine qui se trouve être sur le même sous-réseau que le *Client*. Il est recommandé que les *Clients* ignorent toutes les URL associées au domaine "local" de niveau supérieur, sauf si elles sont renvoyées par le LDS-ME qui s'exécute sur la même machine.

Les administrateurs système peuvent résoudre ce problème en configurant un DNS normal avec les noms de domaine pleinement qualifiés pour toutes les machines auxquelles les *Clients* ont besoin d'accéder en dehors du *MulticastSubnet*.

Il convient que les *Serveurs* configurés avec des noms de domaine pleinement qualifiés spécifient les noms de domaine pleinement qualifiés dans leur *Certificat* d'*ApplicationInstance*. Les *Serveurs* ne doivent pas spécifier de domaine "local" de niveau supérieur dans leur *Certificat*. Les *Clients* qui utilisent une URL renvoyée par un LDS-ME doivent ignorer le domaine "local" de niveau supérieur lorsqu'ils comparent le domaine et le *Certificat du Serveur*.

Noter que la validation du nom de domaine est nécessaire; toutefois, elle ne constitue pas une vérification suffisante contre les *Serveurs* malveillants ou les attaques de l'homme du milieu lorsque les *Certificats* du *Serveur* ne contiennent pas de noms de domaine pleinement qualifiés. La relation de confiance du *Certificat* établi par les administrateurs constitue le mécanisme principal utilisé pour prévenir de tels risques.

6.3 Modèle d'Information

6.3.1 Vue d'ensemble

Le *Modèle d'Information* du *GlobalDiscoveryServer* utilisé pour la *découverte* est représenté à la Figure 11. La plupart des interactions entre le *GlobalDiscoveryServer* et l'administrateur des *Applications* ou le *Client* s'effectuent par le biais des *Méthodes* définies dans le dossier *Répertoire*.

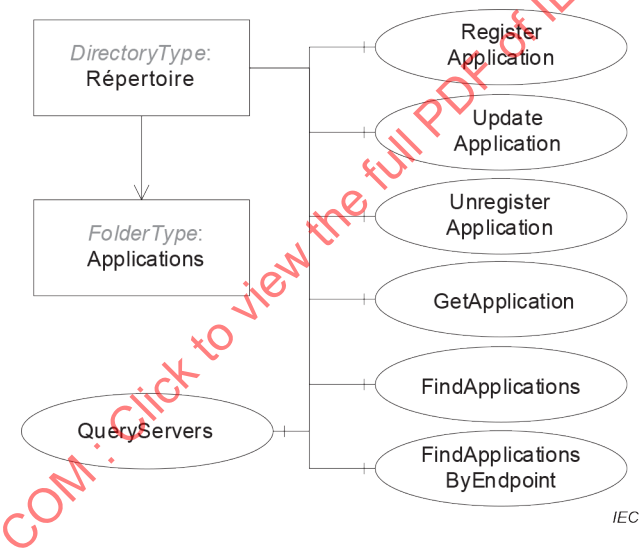


Figure 11 – Espace d'adressage du GDS

6.3.2 Répertoire

Cet *Objet* constitue la racine de l'*AddressSpace* du *GlobalDiscoveryServer* et la cible d'une référence *Organizes* du dossier *Objets* défini dans l'IEC 62541-5. Il organise les informations qui peuvent être accessibles dans les sous-dossiers. La mise en œuvre d'un GDS lui permet de personnaliser et d'organiser les dossiers comme il le souhaite. Par exemple, il peut exister des dossiers pour les modèles d'information, pour les services facultatifs ou pour divers emplacements d'un domaine administratif. Il est défini dans le Tableau 2.

Tableau 2 – Définition de l'Objet Répertoire

Attribut	Valeur				
BrowseName	Répertoire				
Namespace	GDS (voir 3.3)				
TypeDefinition	DirectoryType défini en 6.3.3.				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

6.3.3 DirectoryType

DirectoryType est l'*ObjectType* qui s'applique à la racine de l'*AddressSpace* du *GlobalDiscoveryServer*. Il organise les informations qui peuvent être accessibles dans les sous-dossiers. Il fournit également des méthodes permettant aux applications de s'enregistrer ou de trouver d'autres applications. Il est défini dans le Tableau 3.

Tableau 3 – Définition de DirectoryType

Attribut	**Valeur**				
BrowseName	DirectoryType				
Namespace	GDS (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *FolderType* défini dans l'IEC 62541-5.					
Organizes	Objet	Applications	-	FolderType	Obligatoire
HasComponent	Méthode	FindApplications	Défini en 6.3.4		Obligatoire
HasComponent	Méthode	RegisterApplication	Défini en 6.3.6		Obligatoire
HasComponent	Méthode	UpdateApplication	Défini en 6.3.7		Obligatoire
HasComponent	Méthode	UnregisterApplication	Défini en 6.3.8		Obligatoire
HasComponent	Méthode	GetApplication	Défini en 6.3.9		Obligatoire
HasComponent	Méthode	QueryApplications	Défini en 6.3.10		Obligatoire
HasComponent	Méthode	QueryServers	Défini en 6.3.11		Obligatoire

Le dossier *Applications* peut contenir des *Objets* représentant les *Applications* connues du GDS. Ces *Objets* peuvent être organisés en sous-dossiers comme déterminé par le GDS. Les réseaux, l'emplacement physique ou les profils pris en charge constituent certaines des caractéristiques d'organisation des applications. La *Méthode QueryServers* peut être utilisée pour rechercher des *Applications* OPC UA sur la base de différents critères.

Il n'est pas exigé qu'un GDS expose ses *Applications* en tant qu'*Objets* navigables dans son *AddressSpace*; toutefois, chaque *Application* doit disposer d'un *NodeId* unique qui peut être transmis aux *Méthodes* utilisées pour administrer le GDS.

La *Méthode FindApplications* renvoie les *Applications* associées à un *ApplicationUri*. Elle peut être appelée par n'importe quelle application *Client*.

La *Méthode RegisterApplication* est utilisée pour ajouter une nouvelle *Application* au GDS. Elle exige des privilèges d'administration.

La *Méthode UpdateApplication* est utilisée pour mettre à jour une *Application* existante dans le GDS. Elle exige des privilèges d'administration.

La *Méthode UnregisterApplication* est utilisée pour retirer une *Application* du GDS. Elle exige des privilèges d'administration.

La *Méthode QueryApplications* est utilisée pour trouver des applications *Client* ou *Serveur* qui répondent aux critères fournis. Cette *Méthode* remplace la *Méthode QueryServers*.

La *Méthode QueryServers* est utilisée pour trouver des *Serveurs* qui répondent aux critères spécifiés. Elle peut être appelée par n'importe quelle application *Client*. Cette *Méthode* a été remplacée par la *Méthode QueryApplications*.

6.3.4 FindApplications

La *Méthode FindApplications* est utilisée pour trouver l'*ApplicationId* d'une *Application* OPC UA connue du GDS. En situation normale, la liste des enregistrements renvoyés ne comporte pas plus d'une entrée; toutefois, les erreurs de configuration système peuvent créer une situation dans laquelle le GDS comporte plusieurs entrées pour un *ApplicationUri* unique. Si cela se produit, une personne devra probablement examiner ces enregistrements afin de déterminer lequel d'entre eux correspond véritablement à l'*ApplicationUri*.

Si la matrice renvoyée est nulle ou que sa longueur est de zéro, alors le GDS ne comporte pas d'entrée pour l'*ApplicationUri*.

Signature

```
FindApplications(
      [in]  String applicationUri
      [out] ApplicationRecordDataType[] applications
      );
```

Argument	Description
applicationUri	*ApplicationUri* qui identifie l'*Application* d'intérêt.
applications	Liste des enregistrements d'application qui correspondent à l'*ApplicationUri*. L'ApplicationRecordDataType est défini en 6.3.5.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 4 spécifie la représentation de l'*AddressSpace* pour la *Méthode FindApplications*.

Tableau 4 – Définition de l'AddressSpace pour la Méthode FindApplications

Attribut	Valeur				
BrowseName	FindApplications				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument	PropertyType	Obligatoire

6.3.5 ApplicationRecordDataType

Ce type définit un DataType qui représente un enregistrement dans le GDS (voir Tableau 5).

Tableau 5 – Définition d'ApplicationRecordDataType

Nom	Type	Valeur
applicationId	NodeId	Identificateur unique attribué à l'enregistrement par le GDS. Ce *NodeId* peut être transmis aux autres *Méthodes*.
applicationUri	Chaîne	URI de l'*Application* associée à l'enregistrement.
applicationType	ApplicationType	Type de l'application. Ce type est défini dans l'IEC 62541-4.
applicationNames	LocalizedText	Un ou plusieurs noms localisés pour l'application. Le premier élément est l'*ApplicationName* par défaut de l'application lorsqu'un nom non localisé est nécessaire.
productUri	Chaîne	URI globalement unique pour le produit associé à l'application. Cet URI est attribué par le fournisseur de l'application.
discoveryUrls	Chaîne	Liste des URL de découverte d'une application. Le premier élément est la valeur par défaut si un *Client* a besoin de choisir une URL. La première URL HTTPS spécifie le domaine utilisé comme Nom Commun des *Certificats* HTTPS. Si l'*ApplicationType* est *Client*, toutes les URL doivent comporter le préfixe "inv+" indiquant qu'elles prennent en charge les connexions inverses.
serverCapability Identifiers	Chaîne	Liste des identificateurs des fonctionnalités d'un serveur pour l'application. Les valeurs admises sont définies à l'Annexe D.

6.3.6 RegisterApplication

La *Méthode RegisterApplication* est utilisée pour enregistrer une nouvelle *Instance d'Application* auprès d'un *GlobalDiscoveryServer*.

Cette *Méthode* doit uniquement être invoquée par les utilisateurs autorisés.

Les *Serveurs* qui prennent en charge la redondance transparente doivent s'enregistrer en tant qu'application unique et traverser les *DiscoveryUrls* de toutes les instances et/ou de tous les chemins réseau disponibles.

La *Méthode RegisterApplication* crée des enregistrements en double si l'*ApplicationUri* existe déjà, étant donné qu'une mauvaise configuration des applications peut se traduire par différentes applications ayant le même *ApplicationUri*. Avant d'appeler cette *Méthode*, le *Client* doit appeler la *Méthode FindApplications* pour vérifier s'il existe déjà un enregistrement pour l'application qu'il utilise. Si des enregistrements sont identifiés comme semblant appartenir à différentes applications (les *DiscoveryUrls* sont différentes, par exemple), le *Client* doit alors signaler un avertissement avant de poursuivre.

Si l'enregistrement a réussi et que l'audit est pris en charge, le GDS doit générer l'*ApplicationRegistrationChangedAuditEventType* (voir 6.3.12).

Signature

```
RegisterApplication(
    [in]  ApplicationRecordDataType application
     [out] NodeId applicationId
);
```

Argument	Description
application	Application à enregistrer auprès du *GlobalDiscoveryServer*.
applicationId	Identificateur unique pour l'*Application* enregistrée. Cet identificateur est permanent et est utilisé dans les autres *Méthodes* permettant d'administrer les applications.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_InvalidArgument	L'application ou l'un des champs de l'enregistrement d'application n'est pas valide. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 6 spécifie la représentation de l'*AddressSpace* pour la *Méthode RegisterApplication*.

Tableau 6 – Définition de l'AddressSpace pour la Méthode RegisterApplication

Attribut	Valeur				
BrowseName	RegisterApplication				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument	PropertyType	Obligatoire

6.3.7 UpdateApplication

La *Méthode UpdateApplication* est utilisée pour mettre à jour une *Application* existante dans un *GlobalDiscoveryServer*.

Cette *Méthode* doit uniquement être invoquée par les utilisateurs autorisés.

Si la mise à jour a réussi et que l'audit est pris en charge, le GDS doit générer l'*ApplicationRegistrationChangedAuditEventType* (voir 6.3.12).

Signature

```
UpdateApplication(
      [in]  ApplicationRecordDataType application
    );
```

Argument	Description
application	Application à mettre à jour dans la base de données du GDS.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'applicationId n'est pas connu du GDS.
Bad_InvalidArgument	L'application ou l'un des champs de l'enregistrement d'application n'est pas valide. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 7 spécifie la représentation de l'*AddressSpace* pour la *Méthode UpdateApplication*.

Tableau 7 – Définition de l'AddressSpace pour la Méthode UpdateApplication

Attribut	Valeur				
BrowseName	UpdateApplication				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument	PropertyType	Obligatoire

6.3.8 UnregisterApplication

La *Méthode UnregisterApplication* est utilisée pour retirer une *Application* d'un *GlobalDiscoveryServer.*

Cette *Méthode* doit uniquement être invoquée par les utilisateurs autorisés.

Une *Application Serveur* non enregistrée peut être automatiquement rajoutée si le GDS est configuré de sorte à s'autoalimenter en appelant le Service FindServersOnNetwork et si l'Application Serveur est toujours enregistrée auprès de son LDS local.

Si l'annulation de l'enregistrement a réussi et que l'audit est pris en charge, le GDS doit générer l'*ApplicationRegistrationChangedAuditEventType* (voir 6.3.12).

Signature

```
UnregisterApplication(
      [in] NodeId applicationId
    );
```

Argument	Description
applicationId	Identificateur attribué à l'*Application* par le GDS.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*ApplicationId* n'est pas connu du GDS.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 8 spécifie la représentation de l'*AddressSpace* pour la *Méthode UnregisterApplication*.

Tableau 8 – Définition de l'AddressSpace pour la Méthode UnregisterApplication

Attribut	Valeur				
BrowseName	UnregisterApplication				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument	PropertyType	Obligatoire

6.3.9 GetApplication

La *Méthode GetApplication* est utilisée pour trouver une *Application* OPC UA connue du GDS.

Signature

```
GetApplication(
      [in]  NodeId applicationId
      [out] ApplicationRecordDataType application
      );
```

Argument	Description
applicationId	*ApplicationId* qui identifie l'*Application* d'intérêt.
application	Enregistrement d'application qui correspond à l'*ApplicationId*. L'ApplicationRecordDataType est défini en 6.3.5.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	Aucun enregistrement trouvé pour l'*ApplicationId* spécifié.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 9 spécifie la représentation de l'*AddressSpace* pour la *Méthode GetApplication*.

Tableau 9 – Définition de l'AddressSpace pour la Méthode GetApplication

Attribut	Valeur				
BrowseName	GetApplication				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument	PropertyType	Obligatoire

6.3.10 QueryApplications

La *Méthode QueryApplications* est utilisée pour trouver des applications *Client* ou *Serveur* qui correspondent aux filtres spécifiés. Les seules applications Client renvoyées sont celles qui prennent en charge la fonctionnalité de connexion inverse décrite dans l'IEC 62541-6.

La *Méthode QueryApplications* renvoie les *ApplicationDescriptions* à la place des *Structures ServerOnNetwork* renvoyées par la *Méthode QueryServers*. Cette *Méthode* s'avère plus utile pour certains *Clients*, car elle correspond au type de résultat renvoyé par le *Service FindServers*.

Tout *Client* est capable d'appeler cette *Méthode*; toutefois, l'ensemble des résultats renvoyés peut être limité en fonction des authentifiants d'utilisateur du *Client*.

Les applications renvoyées doivent traverser tous les filtres fournis (autrement dit, les filtres sont combinés dans une opération AND). Le paramètre *capabilities* est une matrice; l'application traverse ce filtre si elle prend en charge toutes les fonctionnalités spécifiées.

À chaque fois que le GDS crée ou met à jour un enregistrement d'application, il doit affecter un identificateur à croissance monotone à l'enregistrement. Cela permet aux *Clients* de demander des enregistrements par lot en spécifiant l'identificateur du dernier enregistrement reçu lors du dernier appel de la *Méthode QueryApplications*. Pour ce faire, le GDS doit renvoyer les enregistrements dans l'ordre, en commençant par l'identificateur d'enregistrement le plus petit. Le GDS doit également renvoyer l'horodatage de la dernière réinitialisation du compteur. Si un *Client* détecte que cet horodatage est plus récent que l'horodatage du dernier appel de la *Méthode*, il doit de nouveau appeler la *Méthode* avec un *startingRecordId* de 0.

Signature

```
QueryApplications(
      [in]  UInt32 startingRecordId
      [in]  UInt32 maxRecordsToReturn
      [in]  String applicationName
      [in]  String applicationUri
      [in]  UInt32 applicationType
      [in]  String productUri
      [in]  String[] capabilities
      [out] DateTime lastCounterResetTime
      [out] UInt32 nextRecordId
      [out] ApplicationDescription[] applications
    );
```

Argument	Description
ENTRÉES	
startingRecordId	Seuls les enregistrements dont l'identificateur est supérieur à ce nombre sont renvoyés. Spécifier 0 pour commencer par le premier enregistrement de la base de données.
maxRecordsToReturn	Nombre maximal d'enregistrements à renvoyer dans la réponse. 0 indique qu'il n'y a pas de limite.
applicationName	*ApplicationName* des applications à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée. Le filtre est uniquement appliqué à l'*ApplicationName* par défaut.
applicationUri	*ApplicationUri* des applications à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée.
applicationType	Masque indiquant les types d'applications qui sont renvoyés. Les valeurs du masque sont les suivantes: 0x1 – Serveurs; 0x2 – Clients. Si le masque est 0, toutes les applications sont renvoyées.
productUri	*ProductUri* des applications à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée.
capabilities	Fonctionnalités prises en charge par les applications renvoyées. Les applications renvoyées doivent prendre en charge toutes les fonctionnalités spécifiées. Si aucune fonctionnalité n'est fournie, ce filtre n'est pas utilisé.
SORTIES	
lastCounterResetTime	Horodatage de la dernière réinitialisation du compteur.
nextRecordId	Identificateur du prochain enregistrement. Il est transmis en tant que *startingRecordId* dans les appels ultérieurs de la *Méthode QueryApplications* pour extraire le prochain lot d'enregistrements. Si l'identificateur est 0, il n'existe aucun autre enregistrement à renvoyer.
applications	Liste des *Applications* qui répondent aux critères. La structure *ApplicationDescription* est définie dans l'IEC 62541-4.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 10 spécifie la représentation de l'*AddressSpace* pour la *Méthode QueryApplications*.

Tableau 10 – Définition de l'AddressSpace pour la Méthode QueryApplications

Attribut	Valeur				
BrowseName	QueryApplications				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

6.3.11 QueryServers (obsolète)

La *Méthode QueryServers* est utilisée pour trouver des applications *Serveur* qui correspondent aux filtres spécifiés.

Tout *Client* est capable d'appeler cette *Méthode*; toutefois, l'ensemble des résultats renvoyés peut être limité en fonction des authentifiants d'utilisateur du *Client*.

Les applications renvoyées doivent traverser tous les filtres fournis (autrement dit, les filtres sont combinés dans une opération AND). Le paramètre *serverCapabilities* est une matrice; l'application traverse ce filtre si elle prend en charge toutes les fonctionnalités spécifiées.

À chaque fois que le GDS crée ou met à jour un enregistrement d'application, il doit affecter un identificateur à croissance monotone à l'enregistrement. Cela permet aux *Clients* de demander des enregistrements par lot en spécifiant l'identificateur du dernier enregistrement reçu lors du dernier appel de la *Méthode QueryServers*. Pour ce faire, le GDS doit renvoyer les enregistrements dans l'ordre, en commençant par l'identificateur d'enregistrement le plus petit. Le GDS doit également renvoyer l'horodatage de la dernière réinitialisation du compteur. Si un *Client* détecte que cet horodatage est plus récent que l'horodatage du dernier appel de la *Méthode*, il doit de nouveau appeler la *Méthode* avec un *startingRecordId* de 0.

Signature

```
QueryServers(
      [in]  UInt32 startingRecordId
      [in]  UInt32 maxRecordsToReturn
      [in]  String applicationName
      [in]  String applicationUri
      [in]  String productUri
      [in]  String[] serverCapabilities
      [out] DateTime lastCounterResetTime
      [out] ServerOnNetwork[] servers
    );
```

Argument	Description
ENTRÉES	
startingRecordId	Seuls les enregistrements dont l'identificateur est supérieur à ce nombre sont renvoyés. Spécifier 0 pour commencer par le premier enregistrement de la base de données.
maxRecordsToReturn	Nombre maximal d'enregistrements à renvoyer dans la réponse. 0 indique qu'il n'y a pas de limite.
applicationName	*ApplicationName* des *Applications* à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée. Le filtre est uniquement appliqué à l'*ApplicationName* par défaut.
applicationUri	*ApplicationUri* des *Serveurs* à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée.
productUri	*ProductUri* des *Serveurs* à renvoyer. Prend en charge la syntaxe utilisée par le *FilterOperator* LIKE décrit dans l'IEC 62541-4. Non utilisé lorsqu'une chaîne vide est spécifiée.
serverCapabilities	Les applications renvoyées doivent prendre en charge toutes les fonctionnalités de serveur spécifiées. Si aucune fonctionnalité de serveur n'est fournie, ce filtre n'est pas utilisé.
SORTIES	
lastCounterResetTime	Horodatage de la dernière réinitialisation du compteur.
servers	Liste des *Serveurs* qui répondent aux critères. La structure *ServerOnNetwork* est définie dans l'IEC 62541-4.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 11 spécifie la représentation de l'*AddressSpace* pour la *Méthode QueryServers*.

Tableau 11 – Définition de l'AddressSpace pour la Méthode QueryServers

Attribut	Valeur				
BrowseName	QueryServers				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

6.3.12 ApplicationRegistrationChangedAuditEventType

Cet événement est généré lorsque les *Méthodes RegisterApplication*, *UpdateApplication* ou *UnregisterApplication* sont appelées.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 12.

Tableau 12 – Définition d'ApplicationRegistrationChangedAuditEventType

Attribut	Valeur				
BrowseName	ApplicationRegistrationChangedAuditEventType				
Namespace	GDS (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini dans l'IEC 62541-5.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateMethodEventType*. Leur sémantique est définie dans l'IEC 62541-5.

7 Vue d'ensemble de la gestion des certificats

7.1 Vue d'ensemble

Les fonctions de gestion des certificats englobent la gestion et la distribution des certificats et les *Listes de Confiance* des Applications OPC UA. Une application qui propose des fonctions de gestion des certificats est appelée *CertificateManager*. Le GDS et le *CertificateManager* sont généralement réunis au sein d'une même application. Les concepts de base relatifs à la gestion des *Certificats* sont décrits dans l'IEC TR 62541-2.

Il existe deux principaux modèles de gestion des *Certificats*: la gestion en flux tiré et la gestion en flux poussé. Avec la gestion en flux tiré, l'application agit comme un *Client* et utilise les *Méthodes* sur le *CertificateManager* pour demander et mettre à jour les *Certificats* et les *Listes de Confiance*. L'application est chargée de garantir que les *Certificats* et les *Listes de Confiance* sont à jour. Avec la gestion en flux poussé, l'application agit comme un *Serveur* et expose les *Méthodes* que le *CertificateManager* peut appeler pour mettre à jour les *Certificats* et les *Listes de Confiance* selon les exigences.

Le GDS est destiné à fonctionner conjointement avec différents services de Gestion des Certificats, comme le Répertoire Actif. Le GDS fournit un modèle d'information OPC UA normalisé que toutes les applications OPC UA peuvent prendre en charge sans avoir besoin de connaître les particularités d'un système de Gestion des Certificats spécifique.

Il convient que le *CertificateManager* prenne en charge les fonctionnalités suivantes:

- approvisionnement (première configuration d'un dispositif/d'une application);
- renouvellement (renouvellement des certificats expirés ou compromis);
- mise à jour des *Listes de Confiance* (y compris des *Listes de Révocation*);
- révocation (retrait d'un dispositif/d'une application du système).

Bien que l'hypothèse généralement retenue soit que les applications *Client* utilisent le modèle en flux tiré et que les applications *Serveur* utilisent le modèle en flux poussé, cette configuration n'est pas exigée.

Pendant l'approvisionnement, le *CertificateManager* doit être en mesure de fonctionner dans un mode où le *Client* est autorisé à se connecter en toute sécurité avec un *Certificat* valide et où les authentifiants d'utilisateur sont utilisés pour déterminer les droits dont dispose un *Client*; il n'est donc plus nécessaire de configurer des *Listes de Confiance* avant la connexion au *CertificateManager* à des fins d'approvisionnement.

Les fournisseurs d'*Applications* peuvent décider de concevoir l'interaction avec le *CertificateManager* comme un composant distinct (partie d'une application d'administration avec accès à la configuration OPC UA de l'*Application*, par exemple). L'opération est transparente pour le *CertificateManager* et n'est pas décrite en détail dans le reste du présent article.

Le présent document ne définit pas comment administrer un *CertificateManager*; toutefois, un *CertificateManager* doit fournir un système intégré offrant une gestion en flux tiré comme en flux poussé.

7.2 Gestion en flux tiré

La gestion en flux tiré s'effectue en utilisant le modèle d'information *CertificateManager* (en particulier ses Méthodes) défini en 7.6. Les interactions entre l'*Application* et le *CertificateManager* dans le cadre de la gestion en flux tiré sont représentées à la Figure 12.

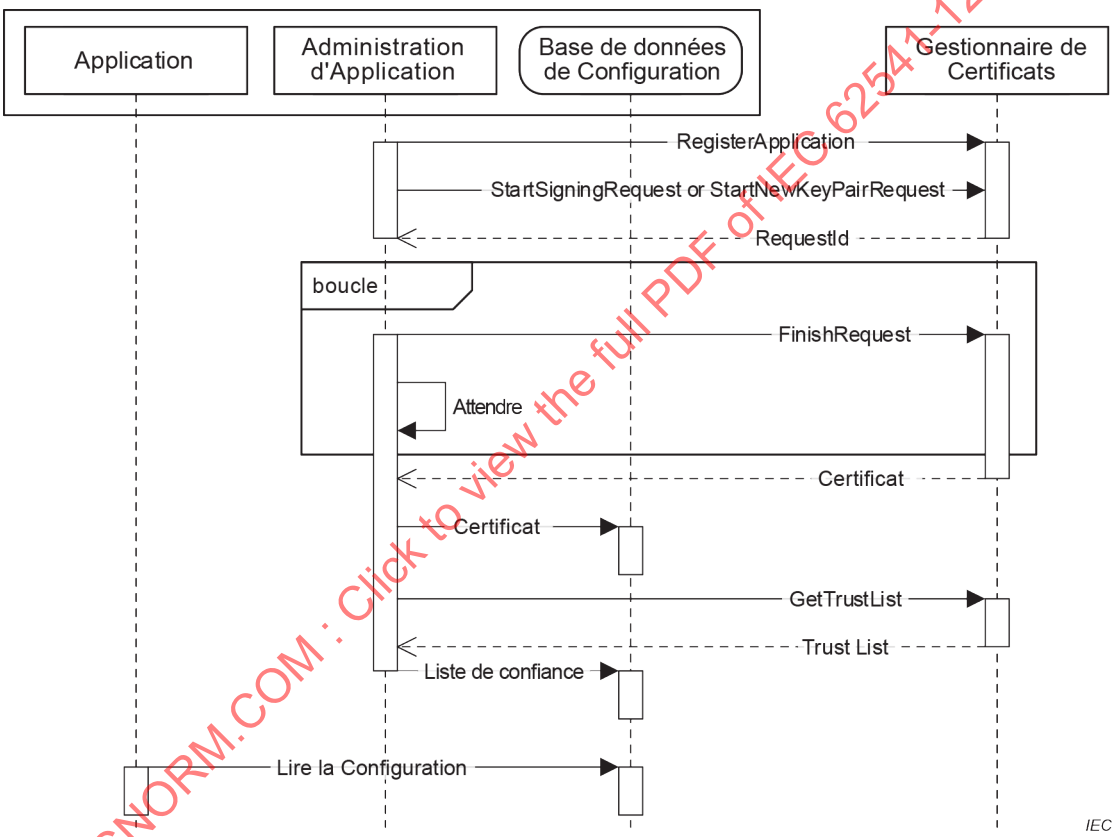


Figure 12 – Modèle de gestion des certificats en flux tiré

Le composant d'Administration d'Application peut faire partie de l'Application ou d'un service autonome qui comprend la manière dont l'Application conserve ses informations de configuration dans sa Base de données de Configuration.

Un processus similaire est utilisé pour renouveler les certificats ou pour mettre régulièrement à jour les *Listes de Confiance*.

Avec le modèle de gestion en flux tiré, la sécurité exige d'utiliser un canal chiffré et les authentifiants d'*Administrateur* du *CertificateManager* afin de garantir que seuls les utilisateurs autorisés peuvent enregistrer de nouvelles *Applications* et demander un nouveau *Certificat* initial. Lorsque l'*Application* dispose d'un *Certificat*, elle peut utiliser ce *Certificat* pour renouveler le *Certificat* ou pour mettre à jour les *Listes de Confiance* et les *Listes de Révocation*. Il est important que le *CertificateManager* n'assure pas le renouvellement des certificats, sauf pour les applications qui possèdent déjà le précédent certificat.

7.3 Gestion en flux poussé

La gestion en flux poussé cible les applications *Serveur*. Elle repose sur les *Méthodes* définies en 7.7 pour obtenir une *Demande* de *Certificat* qui peut être transmise au *CertificateManager.* Après la signature du *Certificat* par le *CertificateManager*, le nouveau *Certificat* est poussé vers le *Serveur* à l'aide de la *Méthode UpdateCertificate*.

Les interactions entre une *Application Serveur* et le *CertificateManager* dans le cadre de la gestion en flux poussé sont représentées à la Figure 13.

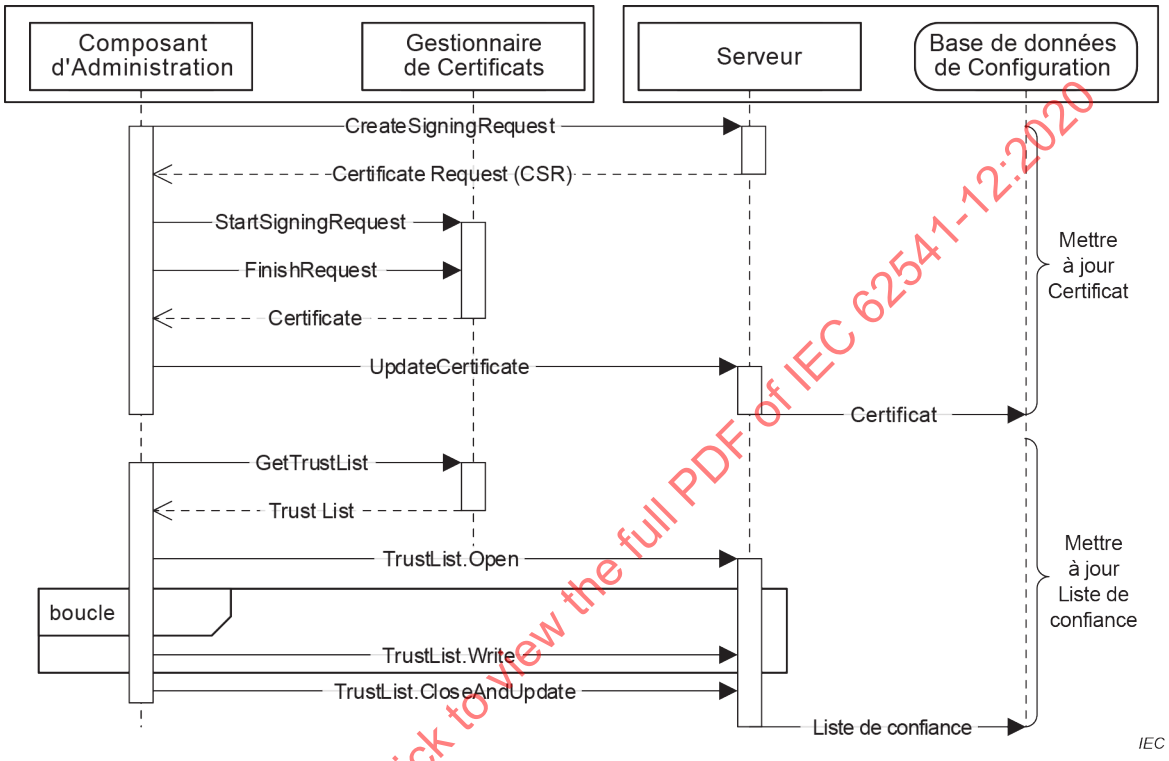


Figure 13 – Modèle de gestion des certificats en flux poussé

Le Composant d'Administration peut faire partie du *CertificateManager* ou d'un service autonome qui utilise OPC UA pour communiquer avec le *CertificateManager* (voir 7.2 pour une description plus complète des interactions exigées pour ce cas d'utilisation). La Base de données de Configuration est utilisée par le *Serveur* pour conserver ses informations de configuration. Il est admis par hypothèse que la *Méthode RegisterApplication* (ou équivalent interne) a été appelée avant que la séquence du schéma ne commence.

Un processus similaire est utilisé pour renouveler les certificats ou pour mettre régulièrement à jour les *Listes de Confiance*.

Avec le modèle de gestion en flux poussé, la sécurité exige d'utiliser un canal chiffré et les authentifiants d'Administrateur du *Serveur* afin de garantir que seuls les utilisateurs autorisés peuvent mettre à jour les *Certificats* ou les *Listes de Confiance*. En outre, des authentifiants d'*Administrateur* distincts sont exigés pour le *CertificateManager*, afin de garantir que seuls les utilisateurs autorisés peuvent enregistrer de nouveaux *Serveurs* et demander de nouveaux *Certificats*.

7.4 Approvisionnement

L'approvisionnement est l'installation initiale d'un *Serveur* ou d'un *Client* OPC UA dans un système où un GDS est disponible et gère tous les certificats. Pour les applications utilisant une interface *Client*, l'approvisionnement peut être réalisé à l'aide d'un Modèle en Flux Tiré. Les applications utilisant une interface *Serveur* peuvent être approvisionnées à l'aide d'un Modèle en Flux Poussé.

En général, les *Serveurs* OPC UA génèrent automatiquement un *Certificat* autosigné lorsqu'ils démarrent pour la première fois. Ils peuvent également disposer d'une *Liste de Confiance* préconfigurée contenant les *Applications* admises pour approvisionner le *Serveur*. Par exemple, les fournisseurs de dispositifs peuvent faire appel à une CA pour délivrer des *Certificats* pour les *Applications* utilisées par leurs techniciens sur site.

Pour les dispositifs intégrés, il convient que le *Serveur* permette à tout *Client* fournissant les authentifiants d'*Administrateur* adéquats de créer la connexion sécurisée nécessaire à l'approvisionnement par gestion en flux poussé. Lorsque le dispositif a obtenu sa *Liste de Confiance* initiale, il convient que le *Serveur* restreigne l'accès aux *Clients* dont les *Certificats* figurent dans la *Liste de Confiance*. Un processus d'approvisionnement propre au fournisseur est exigé si le dispositif ne permet à aucun *Client* de se connecter en toute sécurité à des fins d'approvisionnement.

Voir l'Article G.1 pour obtenir des exemples plus spécifiques d'approvisionnement d'une application.

7.5 Modèle d'information commun

7.5.1 Vue d'ensemble

Le modèle d'information commun définit les types utilisés à la fois dans le Modèle en Flux Tiré et dans le Modèle en Flux Poussé.

7.5.2 TrustListType

Ce type définit un *FileType* qui peut être utilisé pour accéder à une *Liste de Confiance*. Le *TrustListType* est défini de manière formelle dans le Tableau 13.

Le *CertificateManager* utilise ce type pour mettre en œuvre le Modèle en Flux Tiré.

Les *Serveurs* utilisent ce type lors de la mise en œuvre du Modèle en Flux Poussé.

Une instance de *TrustListType* doit restreindre l'accès aux utilisateurs ou aux applications appropriée(s). Il peut s'agir d'un utilisateur administratif de *CertificateManager* qui peut modifier le contenu d'une *Liste de Confiance*, il peut s'agir d'un utilisateur administratif qui lit une *Liste de Confiance* pour la déployer sur l'hôte d'une Application, ou il peut s'agir d'une Application qui peut uniquement accéder à la Liste de Confiance qui lui a été attribuée.

Le fichier de *Liste de Confiance* est un flux à codage UA Binaire contenant une instance de *TrustListDataType* (voir 7.5.7).

La *Méthode Open* ne doit pas prendre en charge des modes autres que Read (0x01) et Write + EraseExisting (0x06).

Lorsqu'un *Client* ouvre le fichier à des fins d'écriture, le *Serveur* ne met pas réellement à jour la *Liste de Confiance* tant que la *Méthode CloseAndUpdate* n'est pas appelée. Il suffit d'appeler la *Méthode Close* pour rejeter les mises à jour. Les masques de bits de la structure *TrustListDataType* permettent au *Client* de mettre uniquement à jour une partie de la *Liste de Confiance*.

Lorsque la *Méthode CloseAndUpdate* est appelée, le *Serveur* valide tous les nouveaux *Certificats* et toutes les nouvelles *CRL*. Si cette validation échoue, la *Liste de Confiance* n'est pas mise à jour et le *Serveur* renvoie le code d'erreur de *Certificat* correspondant (voir IEC 62541-4).

Tableau 13 – Définition de TrustListType

Attribut	Valeur				
BrowseName	TrustListType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *FileType* défini dans l'IEC 62541-5.					
HasProperty	Variable	LastUpdateTime	UtcTime	PropertyType	Obligatoire
HasProperty	Variable	UpdateFrequency	Durée	PropertyType	Facultative
HasComponent	Méthode	OpenWithMasks	Défini en 7.5.3		Facultative
HasComponent	Méthode	CloseAndUpdate	Défini en 7.5.4		Facultative
HasComponent	Méthode	AddCertificate	Défini en 7.5.5		Facultative
HasComponent	Méthode	RemoveCertificate	Défini en 7.5.6		Facultative

La *Propriété LastUpdateTime* indique l'horodatage de la dernière mise à jour de la *Liste de Confiance* par le biais des *Méthodes* de l'*Objet TrustList*. Elle peut être utilisée pour déterminer si un dispositif possède une *Liste de Confiance* à jour ou pour détecter des modifications inattendues. Les modifications hors bande ne sont pas nécessairement signalées par cette valeur.

La *Propriété UpdateFrequency* spécifie la fréquence à laquelle il est nécessaire de rechercher les éventuelles modifications dans la *Liste de Confiance*. Lorsque le *CertificateManager* spécifie cette valeur, il convient que tous les *Clients* qui lisent une copie de la *Liste de Confiance* se connectent au *CertificateManager* et recherchent les mises à jour de la *Liste de Confiance* dans un délai correspondant à 2 fois l'*UpdateFrequency*. Si l'*Objet TrustList* est contenu dans un *Objet ServerConfiguration*, cette valeur spécifie alors la fréquence à laquelle le *Serveur* prévoit une mise à jour de la *Liste de Confiance*.

Lorsque l'audit est pris en charge, le CertificateManager doit générer le *TrustListUpdatedAuditEventType* (voir 7.5.18) si les *Méthodes CloseAndUpdate*, *AddCertificate* ou *RemoveCertificate* sont appelées.

7.5.3 OpenWithMasks

La *Méthode OpenWithMasks* permet à un *Client* de lire uniquement une partie de la *Liste de Confiance*.

Cette *Méthode* peut uniquement être utilisée pour lire la *Liste de Confiance*.

Signature

```
OpenWithMasks(
      [in]  UInt32 masks
      [out] UInt32 fileHandle
    );
```

Argument	Description
masks	Parties de la *Liste de Confiance* incluses dans le fichier à lire. Les masques sont définis en 7.5.8.
fileHandle	Descripteur du fichier récemment ouvert.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 14 spécifie la représentation de l'*AddressSpace* pour la *Méthode OpenWithMasks*.

Tableau 14 – Définition de l'AddressSpace pour la Méthode OpenWithMasks

Attribut	Valeur				
BrowseName	OpenWithMasks				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.5.4 CloseAndUpdate

La *Méthode CloseAndUpdate* ferme le fichier et applique les modifications à la *Liste de Confiance*. Elle peut uniquement être appelée si le fichier a été ouvert pour écriture. Si la *Méthode Close* est appelée, toute donnée mise en cache est éliminée et la *Liste de Confiance* n'est pas modifiée.

Le *Serveur* doit vérifier que chaque *Certificat* de la nouvelle *Liste de Confiance* est valide conformément aux règles obligatoires définies dans l'IEC 62541-4. Lorsqu'un *Certificat* non valide est trouvé, le *Serveur* doit renvoyer une erreur et ne doit pas mettre à jour la *Liste de Confiance*. Si seule une partie de la *Liste de Confiance* est mise à jour, le *Serveur* crée une *Liste de Confiance* temporaire qui inclut la *Liste de Confiance* existante ainsi que les mises à jour, puis valide la *Liste de Confiance* temporaire.

Si le fichier ne peut pas être traité, cette *Méthode* ferme malgré tout le fichier et élimine les données avant de renvoyer une erreur. Cette *Méthode* est exigée si le *Serveur* prend en charge les mises à jour de la *Liste de Confiance*.

La structure chargée inclut un masque (voir 7.5.8) qui spécifie quels champs sont mis à jour. Si un bit n'est pas défini, le champ associé n'est pas modifié.

Signature

```
CloseAndUpdate(
      [in]  UInt32 fileHandle
      [out] Boolean applyChangesRequired
     );
```

Argument	Description
fileHandle	Descripteur du fichier précédemment ouvert.
applyChangesRequired	Fanion indiquant si la *Méthode ApplyChanges* (voir 7.7.5) doit être appelée avant que la *Liste de Confiance* ne soit utilisée par le *Serveur*.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_CertificateInvalid	Le Serveur n'a pas pu valider tous les Certificats de la Liste de Confiance. Les DiagnosticInfo doivent spécifier quels Certificats ne sont pas valides et indiquer l'erreur spécifique.

Le Tableau 15 spécifie la représentation de l'*AddressSpace* pour la *Méthode CloseAndUpdate*.

Tableau 15 – Définition de l'AddressSpace pour la Méthode CloseAndUpdate

Attribut	Valeur				
BrowseName	CloseAndUpdate				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.5.5 AddCertificate

La *Méthode AddCertificate* permet à un *Client* d'ajouter un *Certificat* unique à la *Liste de Confiance*. Le *Serveur* doit vérifier que le *Certificat* est valide conformément aux règles définies dans l'IEC 62541-4. Lorsqu'un *Certificat* non valide est trouvé, le *Serveur* doit renvoyer une erreur et ne doit pas mettre à jour la *Liste de Confiance*.

Cette *Méthode* renvoie une erreur de validation si le *Certificat* est délivré par une CA et que le *Certificat* de l'émetteur ne figure pas dans la *Liste de Confiance*.

Cette *Méthode* ne peut pas fournir de CRL, aussi les certificats d'émetteur ne peuvent pas être ajoutés par cette *Méthode*. A la place, les *Certificats* de CA et leurs CRL doivent être gérés à l'aide de la *Méthode Write* sur l'*Objet TrustList* conteneur.

Cette *Méthode* ne peut pas être appelée si l'*Objet TrustList* conteneur est ouvert.

```
AddCertificate(
    [in] ByteString certificate
    [in] Boolean isTrustedCertificate
);
```

Argument	Description
Certificate	Certificat à codage DER à ajouter.
isTrustedCertificate	Si l'argument est réglé sur TRUE, le Certificat est ajouté à la Liste des Certificats de Confiance. S'il est réglé sur FALSE, le Certificat est ajouté à la Liste des Certificats d'Emetteur.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_CertificateInvalid	Le certificat à ajouter n'est pas valide.
Bad_InvalidState	L'objet est ouvert.

Le Tableau 16 spécifie la représentation de l'*AddressSpace* pour la *Méthode AddCertificate*.

Tableau 16 – Définition de l'AddressSpace pour la Méthode AddCertificate

Attribut	Valeur				
BrowseName	AddCertificate				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire

7.5.6 RemoveCertificate

La *Méthode RemoveCertificate* permet à un *Client* de retirer un *Certificat* unique de la *Liste de Confiance*. Elle renvoie l'erreur *Bad_InvalidArgument* si l'empreinte de signature ne correspond pas à un Certificat de la *Liste de Confiance*.

Si le *Certificat* est un *Certificat* de CA associé à des CRL, toutes les CRL sont également retirées.

Cette méthode ne peut pas être appelée si l'objet dossier est ouvert.

```
RemoveCertificate(
      [in] String thumbprint
      [in] Boolean isTrustedCertificate
    );
```

Argument	Description
Thumbprint	Hachage SHA1 du *Certificat* à retirer.
isTrustedCertificate	Si l'argument est réglé sur TRUE, le Certificat est retiré de la Liste des Certificats de Confiance. S'il est réglé sur FALSE, le Certificat est retiré de la Liste des Certificats d'Emetteur.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_InvalidArgument	Le certificat à retirer est introuvable.
Bad_InvalidState	L'objet est ouvert.

Le Tableau 17 spécifie la représentation de l'*AddressSpace* pour la *Méthode RemoveCertificate*.

Tableau 17 – Définition de l'AddressSpace pour la Méthode RemoveCertificate

Attribut	Valeur				
BrowseName	RemoveCertificate				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire

7.5.7 TrustListDataType

Ce type définit un DataType qui contient la Liste de Confiance d'un *Serveur*. Ses valeurs sont définies dans le Tableau 18.

Tableau 18 – Définition de TrustListDataType

Nom	Type	Valeur
TrustListDataType	structure	
specifiedLists	TrustListMasks	Masque de bits qui indique quelles listes contiennent des informations. L'énumération *TrustListMasks* décrite en 7.5.8 définit les valeurs admises.
trustedCertificates	ByteString[]	Liste des *Applications* et *Certificats* de CA de confiance.
trustedCrls	ByteString[]	CRL pour les *Certificats* figurant dans la liste de *trustedCertificates*.
issuerCertificates	ByteString[]	Liste des *Certificats* de CA nécessaires pour valider les *Certificats*.
issuerCrls	ByteString[]	CRL pour les *Certificats* de CA figurant dans la liste d'*issuerCertificates*.

7.5.8 TrustListMasks

Il s'agit d'un DataType qui définit les valeurs utilisées pour le champ SpecifiedLists du *TrustListDataType*. Ses valeurs sont définies dans le Tableau 19.

Tableau 19 – Valeurs de TrustListMasks

Valeur	Valeur
None_0	Aucun champ n'est renseigné.
TrustedCertificates_1	Les TrustedCertificates sont fournis.
TrustedCrls_2	Les TrustedCrls sont fournies.
IssuerCertificates_4	Les IssuerCertificates sont fournis.
IssuerCrls_8	Les IssuerCrls sont fournies.
All_15	Tous les champs sont renseignés.

7.5.9 TrustListOutOfDateAlarmType

Le *SystemOffNormalAlarmType* est généré par le *Serveur* lorsque la *Propriété UpdateFrequency* expire et que la *Liste de Confiance* n'a pas été mise à jour. Cette alarme revient automatiquement à la normale lorsque la *Liste de Confiance* est mise à jour. Ce type est défini dans le Tableau 20.

Tableau 20 – Définition de TrustListOutOfDateAlarmType

Attribut	Valeur				
BrowseName	TrustListOutOfDateAlarmType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du SystemOffNormalAlarmType défini dans l'IEC 62541-9.					
HasProperty	Variable	TrustListId	NodeId	PropertyType	Obligatoire
HasProperty	Variable	LastUpdateTime	UtcTime	PropertyType	Obligatoire
HasProperty	Variable	Update Frequency	Durée	PropertyType	Obligatoire

La *Propriété TrustListId* spécifie le *NodeId* de l'*Objet TrustList* obsolète.

La *Propriété LastUpdateTime* spécifie l'horodatage de la dernière mise à jour de la *Liste de Confiance*.

La *Propriété UpdateFrequency* spécifie la fréquence à laquelle il est nécessaire de mettre à jour la *Liste de Confiance*.

7.5.10 CertificateGroupType

Ce type est utilisé pour les *Objets* qui représentent des *Groupes de Certificats* dans l'*AddressSpace*. Un *Groupe de Certificats* est un contexte contenant une *Liste de Confiance* et un ou plusieurs *Certificats* qui peuvent être attribués à une *Application*. Ce type permet à une *Application* possédant plusieurs *Listes de Confiance* et/ou *Certificats d'Application* de les exprimer dans son *AddressSpace*. Ce type est défini dans le Tableau 21.

Tableau 21 – Définition de CertificateGroupType

Attribut	Valeur				
BrowseName	CertificateGroupType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini dans l'IEC 62541-5.					
HasComponent	Objet	TrustList	-	TrustListType	Obligatoire
HasProperty	Variable	CertificateTypes	NodeId[]	PropertyType	Obligatoire
HasComponent	Objet	CertificateExpired		Certificate ExpirationAlarm Type	Facultative
HasComponent	Objet	TrustListOutOfDate		TrustListOutOf DateAlarmType	Facultative

L'*Objet TrustList* est la *Liste de Confiance* associée au *Groupe de Certificats*.

La *Propriété CertificateTypes* spécifie les *NodeIds* des *CertificateTypes* qui peuvent être attribués aux *Applications* appartenant au *Groupe de Certificats*. Par exemple, un *Groupe de Certificats* avec le *NodeId* du *RsaMinApplicationCertificateType* (voir 7.5.15) et le *NodeId* du *RsaSha256ApplicationCertificate* (voir 7.5.16) spécifiés permet à une *Application* de disposer d'un *Certificat d'Instance d'Application* pour chaque type. Les types de base abstraits peuvent être utilisés dans cette valeur et indiquer que n'importe quel sous-type est admis. Si cette liste est vide, le *Groupe de Certificats* ne permet alors pas d'attribuer des *Certificats* aux *Applications* (autrement dit, le *Groupe de Certificats* existe pour permettre à la *Liste de Confiance* associée d'être lue ou mise à jour). Tous les *CertificateTypes* d'un *Groupe de Certificats* donné doivent être des sous-types d'un unique type commun, qui doit être *ApplicationCertificateType* ou *HttpsCertificateType*.

L'*Objet CertificateExpired* est une *Alarme* générée lorsque le *Certificat* associé au *CertificateGroup* est sur le point d'expirer. Le *CertificateExpirationAlarmType* est défini dans l'IEC 62541-9.

L'*Objet TrustListOutOfDate* est une *Alarme* générée lorsque la *Liste de Confiance* n'a pas été mise à jour dans la période spécifiée par l'*UpdateFrequency* (voir 7.5.2). Le *TrustListOutOfDateAlarmType* est défini en 7.5.9.

7.5.11 CertificateType

Ce type est un type de base abstrait qui s'applique aux types qui décrivent l'objectif d'un *Certificat*. Ce type est défini dans le Tableau 22.

Tableau 22 – Définition de CertificateType

Attribut	Valeur				
BrowseName	CertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini dans l'IEC 62541-5					
HasSubtype	ObjectType	ApplicationCertificateType	Défini en 7.5.12		
HasSubtype	ObjectType	HttpsCertificateType	Défini en 7.5.13		
HasSubtype	ObjectType	UserCredentialCertificateType	Défini en 7.5.14		

7.5.12 ApplicationCertificateType

Ce type est un type de base abstrait qui s'applique aux types qui décrivent l'objectif d'un *ApplicationInstanceCertificate*. Ce type est défini dans le Tableau 23.

Tableau 23 – Définition d'ApplicationCertificateType

Attribut	Valeur				
BrowseName	ApplicationCertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *CertificateType* défini en 7.5.11					
HasSubtype	ObjectType	RsaMinApplicationCertificateType		Défini en 7.5.15	
HasSubtype	ObjectType	RsaSha256ApplicationCertificateType		Défini en 7.5.16	

7.5.13 HttpsCertificateType

Ce type est utilisé pour décrire les Certificats destinés à être utilisés comme *Certificats* HTTPS. Ce type est défini dans le Tableau 24.

Tableau 24 – Définition de HttpsCertificateType

Attribut	Valeur				
BrowseName	HttpsCertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *CertificateType* défini en 7.5.11.					

7.5.14 UserCredentialCertificateType

Ce type est utilisé pour décrire les Certificats destinés à être utilisés comme authentifiants d'utilisateur. Ce type est défini dans le Tableau 25.

Tableau 25 – Définition de UserCredentialCertificateType

Attribut	Valeur				
BrowseName	UserCredentialCertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *CertificateType* défini en 7.5.11.					

7.5.15 RsaMinApplicationCertificateType

Ce type est utilisé pour décrire les *Certificats* destinés à être utilisés comme *ApplicationInstanceCertificate*. Ceux-ci doivent disposer d'une clé RSA de 1 024 bits ou 2 048 bits. Toutes les *Applications* qui prennent en charge les profils *Basic128Rsa15* et *Basic256* (voir IEC 62541-7) doivent disposer d'un *Certificat* de ce type. Ce type est défini dans le Tableau 26.

Tableau 26 – Définition de RsaMinApplicationCertificateType

Attribut	Valeur				
BrowseName	RsaMinApplicationCertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 7.5.12.					

7.5.16 RsaSha256ApplicationCertificateType

Ce type est utilisé pour décrire les *Certificats* destinés à être utilisés comme *ApplicationInstanceCertificate*. Ceux-ci doivent disposer d'une clé RSA de 2 048 bits, 3 072 bits ou 4 096 bits. Toutes les *Applications* qui prennent en charge le profil *Basic256Sha256* (voir IEC 62541-7) doivent disposer d'un *Certificat* de ce type. Ce type est défini dans le Tableau 27.

Tableau 27 – Définition de RsaSha256ApplicationCertificateType

Attribut	Valeur				
BrowseName	RsaSha256ApplicationCertificateType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseObjectType* défini en 7.5.12.					

7.5.17 CertificateGroupFolderType

Ce type est utilisé pour les *Dossiers* qui organisent les *Groupes de Certificats* dans l'*AddressSpace*. Ce type est défini dans le Tableau 28.

Tableau 28 – Définition de CertificateGroupFolderType

Attribut	**Valeur**				
BrowseName	CertificateGroupFolderType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *FolderType* défini dans l'IEC 62541-5.					
Organizes	Objet	DefaultApplicationGroup		CertificateGroupType	Obligatoire
Organizes	Objet	DefaultHttpsGroup		CertificateGroupType	Facultative
Organizes	Objet	DefaultUserTokenGroup		CertificateGroupType	Facultative
Organizes	Objet	<AdditionalGroup>		CertificateGroupType	Optional Placeholder

L'*Objet DefaultApplicationGroup* représente le *Groupe de Certificats* par défaut pour les *Applications*. Il est utilisé pour accéder à la *Liste de Confiance des Applications* par défaut et pour définir les *CertificateTypes* admis pour l'*ApplicationInstanceCertificate*. Cet *Objet* doit spécifier le *NodeId* de l'*ApplicationCertificateType* (voir 7.5.12) comme une entrée unique de la liste de *CertificateTypes* ou il doit spécifier un ou plusieurs sous-types de l'*ApplicationCertificateType*.

L'*Objet DefaultHttpsGroup* représente le *Groupe de Certificats* par défaut pour les communications HTTPS. Il est utilisé pour accéder à la *Liste de Confiance* HTTPS par défaut et pour définir les *CertificateTypes* admis pour le *Certificat HTTPS*. Cet *Objet* doit spécifier le *NodeId* du *HttpsCertificateType* (voir 7.5.13) comme une entrée unique de la liste de *CertificateTypes* ou il doit spécifier un ou plusieurs sous-types du *HttpsCertificateType*.

L'*Objet DefaultUserTokenGroup* représente le *Groupe de Certificats* par défaut pour la validation des authentifiants d'utilisateur. Il est utilisé pour accéder à la *Liste de Confiance* des authentifiants d'utilisateur par défaut et pour définir les *CertificateTypes* admis pour le *Certificat* des authentifiants d'utilisateur Cet *Objet* doit laisser la liste de *CertificateTypes* vide.

7.5.18 TrustListUpdatedAuditEventType

Cet événement est généré lorsqu'une *Liste de Confiance* est modifiée.

Il est le résultat de l'appel d'une *Méthode CloseAndUpdate* sur un *Objet TrustListType*.

Il doit également être généré lorsque la *Méthode AddCertificate* ou *RemoveCertificate* provoque une mise à jour de la *Liste de Confiance*.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 29.

Tableau 29 – Définition de TrustListUpdatedAuditEventType

Attribut	**Valeur**				
BrowseName	TrustListUpdatedAuditEventType				
Namespace	CORE (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini dans l'IEC 62541-5.					

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateMethodEventType*. Leur sémantique est définie dans l'IEC 62541-5.

7.6 Modèle d'information pour la gestion des certificats en flux tiré

7.6.1 Vue d'ensemble

L'*AddressSpace* du *GlobalDiscoveryServer* utilisé pour la gestion des *Certificats* est représenté à la Figure 14. La plupart des interactions entre le *GlobalDiscoveryServer* et l'administrateur des *Applications* ou le *Client* s'effectuent par le biais des *Méthodes* définies dans le dossier *Répertoire*.

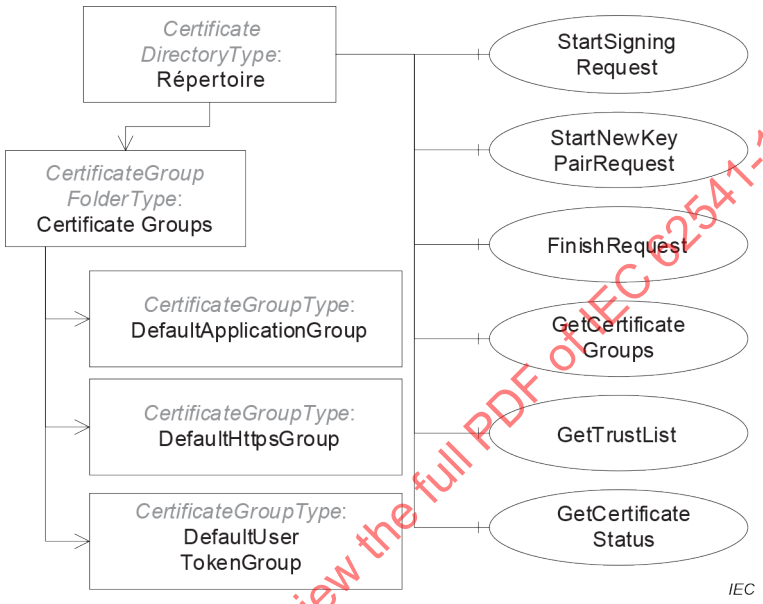


Figure 14 – AddressSpace de gestion des certificats pour le GlobalDiscoveryServer

7.6.2 CertificateDirectoryType

Cet *ObjectType* est la *TypeDefinition* qui s'applique à la racine de l'*AddressSpace* du *CertificateManager*. Il fournit des *Méthodes* supplémentaires pour la gestion des *Certificats*, qui sont décrites dans le Tableau 30.

Tableau 30 – Définition de l'ObjectType CertificateDirectoryType

Attribut	Valeur				
BrowseName	CertificateDirectoryType				
Namespace	GDS (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**Data Type**	**TypeDefinition**	**Modelling Rule**
Sous-type du *DirectoryType* défini en 6.3.3.					
Organizes	Objet	CertificateGroups		CertificateGroup FolderType	Obligatoire
HasComponent	Méthode	StartSigningRequest	Défini en 7.6.3		Obligatoire
HasComponent	Méthode	StartNewKeyPairRequest	Défini en 7.6.4		Obligatoire
HasComponent	Méthode	FinishRequest	Défini en 7.6.5		Obligatoire
HasComponent	Méthode	GetCertificateGroups	Défini en 7.6.6		Obligatoire
HasComponent	Méthode	GetTrustList	Défini en 7.6.6		Obligatoire
HasComponent	Méthode	GetCertificateStatus	Défini en 7.6.8		Obligatoire

L'*Objet CertificateGroups* organise les *Groupes de Certificats* pris en charge par le *CertificateManager*. Il est décrit en 7.5.17. Les *CertificateManagers* doivent prendre en charge le *DefaultApplicationGroup* et peuvent prendre en charge le *DefaultHttpsGroup* ou le *DefaultUserTokenGroup*. Les *CertificateManagers* peuvent prendre en charge des *Groupes de Certificats* supplémentaires en fonction de leurs exigences. Par exemple, un *CertificateManager* avec plusieurs Autorités de Certification représenterait chacune d'elles par un *Objet CertificateGroupType* organisé par le *Dossier CertificateGroups*. Les *Clients* pourraient alors demander des *Certificats* délivrés par une CA spécifique en transmettant le *NodeId* approprié aux *Méthodes StartSigningRequest* ou *StartNewKeyPairRequest*.

La *Méthode StartSigningRequest* est utilisée pour demander un nouveau *Certificat* signé par une CA gérée par le *CertificateManager*. Cette *Méthode* est recommandée lorsque l'appelant dispose déjà d'une clé privée.

La *Méthode StartNewKeyPairRequest* est utilisée pour demander un nouveau *Certificat* signé par une CA gérée par le *CertificateManager*, ainsi qu'une nouvelle clé privée. Cette *Méthode* est uniquement utilisée lorsque l'appelant ne dispose pas d'une clé privée et ne peut pas en générer.

La *Méthode FinishRequest* est utilisée pour vérifier qu'une demande de *Certificat* a été approuvée par l'*Administrateur* du *CertificateManager*. Lorsque l'opération réussit, le *Certificat* et la *Clé Privée* (si demandée) sont renvoyés.

La *Méthode GetCertificateGroups* renvoie une liste de *NodeIds* pour les *Objets CertificateGroupType* pouvant être utilisés pour demander des *Certificats* ou des *Listes de Confiance* pour une *Application*.

La *Méthode GetTrustList* renvoie le *NodeId* d'un *Objet TrustListType* pouvant être utilisé pour lire la *Liste de Confiance* d'une *Application*.

La *Méthode GetCertificateStatus* vérifie s'il est nécessaire que l'*Application* mette à jour son *Certificat*.

7.6.3 StartSigningRequest

StartSigningRequest est utilisé pour initier une demande de création d'un *Certificat* qui utilise la clé privée dont dispose l'appelant sur le moment. Le nouveau *Certificat* est renvoyé dans la réponse *FinishRequest*.

Signature

```
StartSigningRequest(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [in]  NodeId certificateTypeId
      [in]  ByteString certificateRequest
      [out] NodeId requestId
    );
```

Argument	Description
applicationId	Identificateur attribué à l'enregistrement d'*Application* par le *CertificateManager*.
certificateGroupId	*NodeId* du Groupe de Certificats qui fournit le contexte de la nouvelle demande. S'il est nul, le *CertificateManager* doit choisir le *DefaultApplicationGroup*.
certificateTypeId	*NodeId* du *CertificateType* pour le nouveau *Certificat*. S'il est nul, le *CertificateManager* doit générer un *Certificat* fondé sur la valeur de l'argument certificateGroupId.
certificateRequest	La *CertificateRequest* est utilisée pour prouver la possession de la *Clé Privée*. Il s'agit d'un objet blob à codage PKCS #10 au format DER. Si la *CertificateRequest* porte sur un *Certificat d'ApplicationInstance*, elle doit inclure tous les champs exigés par l'IEC 62541-6, comme le *subjectAltName*.
requestId	*NodeId* qui représente la demande. Cette valeur est transmise à la *Méthode FinishRequest*.

L'appel renvoie le *NodeId* transmis à la *Méthode FinishRequest*.

Le paramètre *certificateGroupId* permet à l'appelant de spécifier un *Groupe de Certificats* qui fournit le contexte de la demande. S'il est nul, le *CertificateManager* doit choisir le *DefaultApplicationGroup*. L'ensemble des *Groupes de Certificats* disponibles figure dans le dossier *CertificateGroups* décrit en 7.6.2. Les *Groupes de Certificats* admis pour une *Application* sont renvoyés par la *Méthode GetCertificateGroups* (voir 7.6.6).

Le paramètre *certificateTypeId* spécifie le type de *Certificat* à renvoyer. Les valeurs admises sont spécifiées par la Propriété CertificateTypes de l'Objet spécifié par le paramètre certificateGroupId.

Le paramètre *certificateRequest* est une *Demande de Certificat* à codage DER. Le nom de sujet, le nom de sujet de remplacement ainsi que la clé publique sont copiés dans le nouveau *Certificat*.

Si le *certificateTypeId* est un sous-type de l'*ApplicationCertificateType*, le nom de sujet doit comporter un champ organisation (O=) ou nom de domaine (DC=). La longueur de la clé publique doit satisfaire aux restrictions de longueur du *CertificateType*. Si le *certificateType* est un sous-type du *HttpsCertificateType*, le nom commun du *Certificat* (CN=) doit être identique à un domaine d'une *DiscoveryUrl* qui utilise HTTPS et le nom de sujet doit comporter un champ organisation (O=). La longueur de la clé publique doit être supérieure ou égale à 1 024 bits.

L'*ApplicationUri* doit être spécifié dans la CSR. Le *CertificateManager* doit renvoyer l'erreur *Bad_CertificateUriInvalid* si l'*ApplicationUri* stocké pour l'Application est différent du contenu de la CSR.

Pour les *Serveurs*, la liste des noms de domaine doit être spécifiée dans la CSR. Les domaines doivent inclure le(s) domaine(s) des *DiscoveryUrls* connues du *CertificateManager*.

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires. Elle peut également être invoquée par l'*Application* qui possède la clé privée utilisée pour signer la *CertificateRequest* (par exemple, la clé privée doit être celle utilisée pour créer le *SecureChannel*).

Si l'audit est pris en charge, le *CertificateManager* doit générer le *CertificateRequestedAuditEventType* (voir 7.6.9) lorsque cette *Méthode* réussit ou échoue.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_InvalidArgument	Le certificateGroupId, le *certificateTypeId* ou la *certificateRequest* n'est pas valide. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_RequestNotAllowed	La configuration actuelle du *CertificateManager* n'autorise pas la demande. Il convient que le texte associé à l'erreur en indique la raison exacte.
Bad_CertificateUriInvalid	L'ApplicationUri n'a pas été spécifié dans la CSR ou ne correspond pas à l'enregistrement d'Application.
Bad_NotSupported	L'algorithme de signature, l'algorithme public ou la taille de la clé publique ne sont pas pris en charge par le *CertificateManager*. Le texte associé à l'erreur doit indiquer le problème exact.

Le Tableau 31 spécifie la représentation de l'*AddressSpace* pour la *Méthode StartSigningRequest*.

Tableau 31 – Définition de l'AddressSpace pour la Méthode StartSigningRequest

Attribut	Valeur				
BrowseName	StartSigningRequest				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.4 StartNewKeyPairRequest

Cette *Méthode* est utilisée pour initier une demande de nouveau *Certificat* et de *Clé Privée*. Le *Certificat* et la clé privée sont renvoyés dans la réponse *FinishRequest*.

Signature

```
StartNewKeyPairRequest(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [in]  NodeId certificateTypeId
      [in]  String subjectName
      [in]  String[] domainNames
      [in]  String privateKeyFormat
      [in]  String privateKeyPassword
       [out] NodeId requestId
    );
```

Argument	Description
applicationId	Identificateur attribué à l'*Instance d'Application* par le *CertificateManager*.
certificateGroupId	*NodeId* du Groupe de Certificats qui fournit le contexte de la nouvelle demande. S'il est nul, le *CertificateManager* doit choisir le *DefaultApplicationGroup*.
certificateTypeId	*NodeId* du *CertificateType* pour le nouveau *Certificat*. S'il est nul, le *CertificateManager* doit générer un *Certificat* fondé sur la valeur de l'argument certificateGroupId.
subjectName	Nom de sujet à utiliser pour le *Certificat*. S'il n'est pas spécifié, l'*ApplicationName* et/ou les *domainNames* sont utilisés pour créer une valeur par défaut adaptée. Le format du nom de sujet est une séquence de paires nom-valeur séparées par le symbole "/". Le nom doit être "CN", "O", "OU", "DC", "L", "S" ou "C" et doit être suivi du signe "=", puis de la valeur. Cette valeur peut être tout caractère affichable, sauf ". Si la valeur comporte le symbole "/" ou le signe "=", celui-ci doit être placé entre guillemets doubles (").
domainNames	Noms de domaine à utiliser pour le *Certificat*. S'ils ne sont pas spécifiés, les *DiscoveryUrls* sont utilisées pour créer une valeur par défaut adaptée.
privateKeyFormat	Format de la clé privée. Les valeurs suivantes sont toujours prises en charge: PFX - Codage PKCS #12 PEM - Codage DER base64 (voir IETF RFC 5958).
privateKeyPassword	Mot de passe à utiliser pour la clé privée.
requestId	*NodeId* qui représente la demande. Cette valeur est transmise à la *Méthode FinishRequest*.

L'appel renvoie le *NodeId* transmis à la *Méthode FinishRequest*.

Le paramètre *certificateGroupId* permet à l'appelant de spécifier un *Groupe de Certificats* qui fournit le contexte de la demande. S'il est nul, le *CertificateManager* doit choisir le *DefaultApplicationGroup*. L'ensemble des *Groupes de Certificats* disponibles figure dans le dossier *CertificateGroups* décrit en 7.6.2. Les *Groupes de Certificats* admis pour une *Application* sont renvoyés par la *Méthode GetCertificateGroups* (voir 7.6.6).

Le paramètre *certificateTypeId* spécifie le type de *Certificat* à renvoyer. Les valeurs admises sont spécifiées par la *Propriété CertificateTypes* de l'*Objet* spécifié par le paramètre certificateGroupId.

Le paramètre *subjectName* est une séquence de paires nom-valeur X.500 séparées par le symbole "/". Par exemple: CN=ApplicationName/OU=Group/O=Company.

Si le *certificateType* est un sous-type de l'*ApplicationCertificateType*, le nom de sujet du *Certificat* doit comporter un champ organisation (O=) ou nom de domaine (DC=). La longueur de la clé publique doit satisfaire aux restrictions de longueur du *CertificateType*. Le champ du nom de domaine spécifié dans le nom de sujet est un domaine logique utilisé pour qualifier le nom de sujet qui peut être, ou non, identique au domaine ou à l'adresse IP du champ subjectAltName du *Certificat*.

Si le *certificateType* est un sous-type du *HttpsCertificateType*, le nom commun du *Certificat* (CN=) doit être identique à un domaine d'une *DiscoveryUrl* qui utilise HTTPS et le nom de sujet doit comporter un champ organisation (O=).

Si le champ subjectName est vide ou nul, le *CertificateManager* génère une valeur par défaut adaptée.

Le paramètre *domainNames* est une liste de domaines à inclure dans le *Certificat*. S'il est nul ou vide, le GDS utilise les *DiscoveryUrls* du *Serveur* pour créer une liste. Pour les *Clients*, les *domainNames* sont omis du *Certificat* s'ils ne sont pas fournis de manière explicite.

Le *privateKeyFormat* spécifie le format de la clé privée renvoyée. Toutes les mises en œuvre du *CertificateManager* doivent prendre en charge les formats "PEM" et "PFX".

Le *privateKeyPassword* spécifie le mot de passe de la clé privée. Le *CertificateManager* ne doit pas conserver ces informations et doit les éliminer dès que la nouvelle clé privée est générée.

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires.

Si l'audit est pris en charge, le *CertificateManager* doit générer le *CertificateRequestedAuditEventType* (voir 7.6.9) lorsque cette *Méthode* réussit ou échoue.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NodeIdUnknown	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_InvalidArgument	Les paramètres certificateGroupId, *certificateTypeId*, *subjectName*, *domainNames* ou *privateKeyFormat* ne sont pas valides. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_RequestNotAllowed	La configuration actuelle du CertificateManager n'autorise pas la demande. Il convient que le texte associé à l'erreur en indique la raison exacte.

Le Tableau 32 spécifie la représentation de l'*AddressSpace* pour la *Méthode StartNewKeyPairRequest*.

Tableau 32 – Définition de l'AddressSpace pour la Méthode StartNewKeyPairRequest

Attribut	Valeur				
BrowseName	StartNewKeyPairRequest				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.5 FinishRequest

La *Méthode FinishRequest* est utilisée pour mettre fin à une demande de certificat initiée par un appel de la *Méthode StartNewKeyPairRequest* ou *StartSigningRequest*.

Signature

```
FinishRequest (
      [in]  NodeId applicationId
      [in]  NodeId requestId
      [out] ByteString certificate
      [out] ByteString privateKey
      [out] ByteString[] issuerCertificates
    );
```

Argument	Description
applicationId	Identificateur attribué à l'*Instance d'Application* par le GDS.
requestId	*NodeId* renvoyé par la *Méthode StartNewKeyPairRequest* ou *StartSigningRequest*.
certificate	*Certificat* à codage DER.
privateKey	Clé privée codée au format demandé. Si un mot de passe a été fourni, le blob est protégé par celui-ci. Ce champ est nul lorsqu'aucune clé privée n'a été demandée.
issuerCertificates	*Certificats* exigés pour valider le nouveau *Certificat*.

Cet appel transmet le *NodeId* renvoyé par un appel précédent de la *Méthode StartNewKeyPairRequest* ou *StartSigningRequest*.

Il est prévu qu'un *Client* appelle régulièrement cette *Méthode* jusqu'à ce que le GDS ait approuvé la demande.

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires. Elle peut également être invoquée par l'*Application* qui possède le *Certificat* (par exemple, la clé privée utilisée pour créer le canal doit être la même que celle utilisée pour signer la demande transmise à la *Méthode StartSigningRequest*).

La *Méthode* doit uniquement être appelée par le biais d'un *SecureChannel* pour lequel le chiffrement a été activé.

Si l'audit est pris en charge, le GDS doit générer le *CertificateDeliveredAuditEventType* (voir 7.6.10) lorsque cette *Méthode* réussit ou lorsqu'elle échoue avec tout sauf *Bad_NothingToDo*.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_InvalidArgument	Le *requestId* ne fait pas référence à une demande valide pour l'*Application*.
Bad_NothingToDo	Il n'y a rien à faire, car la demande n'est pas encore terminée.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_RequestNotAllowed	Le *CertificateManager* a rejeté la demande. Il convient que le texte associé à l'erreur en indique la raison exacte.

Le Tableau 33 spécifie la représentation de l'*AddressSpace* pour la *Méthode FinishRequest*.

Tableau 33 – Définition de l'AddressSpace pour la Méthode FinishRequest

Attribut	Valeur				
BrowseName	FinishRequest				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.6 GetCertificateGroups

La *Méthode GetCertificateGroups* renvoie les Groupes de Certificats attribués à l'*Application*.

Signature

```
GetCertificateGroups(
      [in]  NodeId   applicationId
      [out] NodeId[] certificateGroupIds
      );
```

Argument	Description
applicationId	Identificateur attribué à l'*Application* par le GDS.
certificateGroupIds	Identificateur des Groupes de Certificats attribué à l'Application.

Un *Groupe de Certificats* fournit une *Liste de Confiance* et un ou plusieurs *CertificateTypes* qui peuvent être attribués à une *Application*. Les valeurs renvoyées par cette Méthode sont transmises aux *Méthodes GetTrustList* (voir 7.6.7), *StartSigningRequest* (voir 7.6.3) ou *StartNewKeyPairRequest* (voir 7.6.4).

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires. Elle peut également être invoquée par l'*Application* identifiée par l'*applicationId* (par exemple, la clé privée utilisée pour créer le canal doit être celle associée au *Certificat* attribué à l'*Application*).

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 34 spécifie la représentation de l'*AddressSpace* pour la *Méthode GetCertificateGroups*.

Tableau 34 – Définition de l'AddressSpace pour la Méthode GetCertificateGroups

Attribut	Valeur				
BrowseName	GetCertificateGroups				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.7 GetTrustList

La *Méthode GetTrustList* est utilisée pour récupérer le *NodeId* d'une *Liste de Confiance* attribuée à une *Application*.

Signature

```
GetTrustList(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId
      [out] NodeId trustListId
      );
```

Argument	Description
applicationId	Identificateur attribué à l'*Application* par le GDS.
certificateGroupId	Identificateur d'un *Groupe de Certificats* auquel l'*Application* appartient. S'il est nul, le *CertificateManager* doit renvoyer le *trustListId* d'un groupe par défaut adapté pour l'Application.
trustListId	*NodeId* d'un *Objet TrustList* qui peut être utilisé pour télécharger la *Liste de Confiance* attribuée à l'*Application*.

Les autorisations d'accès s'appliquent également aux *Objets TrustList* qui sont renvoyés par cette *Méthode*. Cette *Liste de Confiance* inclut les *Listes de Révocation de Certificat* (CRL) associées aux *Certificats* d'émetteur dans la *Liste de Confiance*.

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires. Elle peut également être invoquée par l'*Application* identifiée par l'*applicationId* (par exemple, la clé privée utilisée pour créer le canal doit être celle associée au *Certificat* attribué à l'*Application*).

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_InvalidArgument	Le paramètre certificateGroupId n'est pas valide. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 35 spécifie la représentation de l'*AddressSpace* pour la *Méthode GetTrustList*.

Tableau 35 – Définition de l'AddressSpace pour la Méthode GetTrustList

Attribut	Valeur				
BrowseName	GetTrustList				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.8 GetCertificateStatus

La *Méthode GetCertificateStatus* est utilisée pour vérifier s'il est nécessaire qu'une *Application* mette à jour son *Certificat*.

Signature

```
GetCertificateStatus(
      [in]  NodeId applicationId
      [in]  NodeId certificateGroupId

      [in]  NodeId certificateTypeId
      [out] Boolean updateRequired
      );
```

Argument	Description
applicationId	Identificateur attribué à l'*Instance d'Application* par le GDS.
certificateGroupId	*NodeId* du Groupe de Certificats qui fournit le contexte. S'il est nul, le *CertificateManager* doit choisir le *DefaultApplicationGroup*.
certificateTypeId	*NodeId* du *CertificateType* pour le *Certificat*. S'il est nul, le *CertificateManager* doit choisir un *Certificat* fondé sur la valeur de l'argument certificateGroupId.
updateRequired	TRUE, si l'*Application* a besoin de demander un nouveau *Certificat* auprès du GDS. FALSE, si l'*Application* peut continuer à utiliser le *Certificat* existant.

Les autorisations d'accès qui s'appliquent à la *Méthode CreateSigningRequest* doivent s'appliquer à la présente *Méthode*.

Cette *Méthode* peut être invoquée par l'outil de configuration qui a fourni les authentifiants d'utilisateur avec les autorisations d'accès nécessaires. Elle peut également être invoquée par l'*Application* identifiée par l'*applicationId* (par exemple, la clé privée utilisée pour créer le canal doit être celle associée au *Certificat* attribué à l'*Application*).

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_NotFound	L'*applicationId* ne fait pas référence à une *Application* enregistrée.
Bad_InvalidArgument	Le paramètre *certificateGroupId* ou *certificateTypeId* n'est pas valide. Le texte associé à l'erreur doit indiquer le problème exact.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 36 spécifie la représentation de l'*AddressSpace* pour la *Méthode GetCertificateStatus*.

Tableau 36 – Définition de l'AddressSpace pour la Méthode GetCertificateStatus

Attribut	Valeur				
BrowseName	GetCertificateStatus				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.9 CertificateRequestedAuditEventType

Cet événement est généré lorsqu'une nouvelle demande de certificat a été acceptée ou rejetée par le GDS.

Il peut être le résultat de l'appel d'une *Méthode StartNewKeyPairRequest* ou *StartSigningRequest*.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 37.

Tableau 37 – Définition de CertificateRequestedAuditEventType

Attribut	Valeur				
BrowseName	CertificateRequestedAuditEventType				
Namespace	GDS (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini dans l'IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Obligatoire
HasProperty	Variable	CertificateType	NodeId	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateMethodEventType*. Leur sémantique est définie dans l'IEC 62541-5.

La *Propriété CertificateGroup* spécifie le *Groupe de Certificats* affecté par la mise à jour.

La *Propriété CertificateType* spécifie le type de *Certificat* mis à jour.

7.6.10 CertificateDeliveredAuditEventType

Cet événement est généré lorsque le GDS délivre un certificat à un *Client*.

Il est le résultat de l'exécution réussie d'une *Méthode FinishRequest*.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 38.

Tableau 38 – Définition de CertificateDeliveredAuditEventType

Attribut	Valeur				
BrowseName	CertificateDeliveredAuditEventType				
Namespace	GDS (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini dans l'IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Obligatoire
HasProperty	Variable	CertificateType	NodeId	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateMethodEventType*. Leur sémantique est définie dans l'IEC 62541-5.

La *Propriété CertificateGroup* spécifie le *Groupe de Certificats* affecté par la mise à jour.

La *Propriété CertificateType* spécifie le type de *Certificat* mis à jour.

7.7 Modèle d'information pour la gestion des certificats en flux poussé

7.7.1 Vue d'ensemble

Si un *Serveur* prend en charge la gestion en flux poussé, il est exigé qu'il prenne en charge un modèle d'information dans le cadre de son espace d'adressage. Il doit prendre en charge l'*Objet ServerConfiguration* représenté à la Figure 15. Cet *Objet* doit uniquement être visible et accessible aux administrateurs et/ou au GDS.

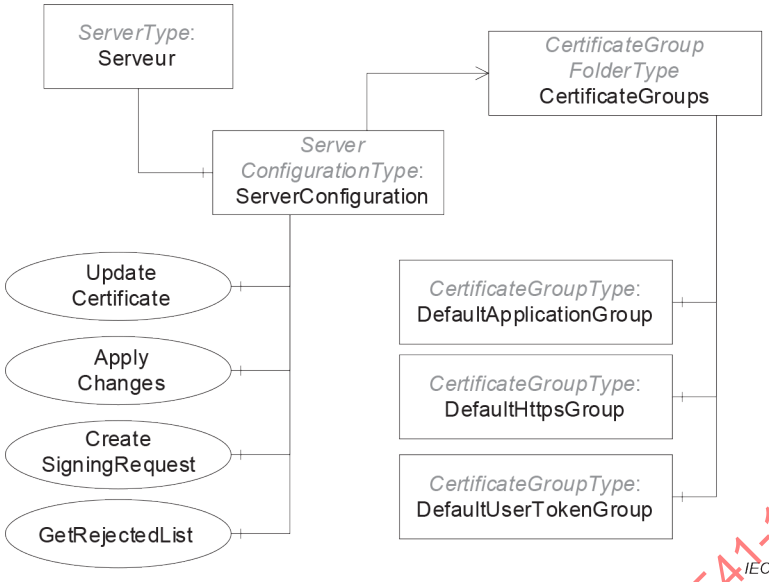


Figure 15 – AddressSpace pour le Serveur prenant en charge la gestion en flux poussé

Tous les accès aux *Méthodes* définies sur l'*Objet ServerConfiguration* doivent s'effectuer par l'intermédiaire d'un canal chiffré. En outre, il convient que les Serveurs disposent d'authentifiants d'utilisateur avec privilèges d'administrateur.

7.7.2 ServerConfiguration

Cet *Objet* permet d'accéder à la configuration du *Serveur* et constitue la cible d'une référence *HasComponent* de l'*Objet Serveur* défini dans l'IEC 62541-5.

Cet *Objet* et ses enfants immédiats doivent être visibles (un droit de navigation est disponible) aux utilisateurs qui peuvent accéder à l'*Objet Serveur*. Il convient que les enfants de l'*Objet CertificateGroups* soient uniquement visibles aux administrateurs autorisés.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 39.

Tableau 39 – Définition de l'Objet ServerConfiguration

Attribut	Valeur				
BrowseName	ServerConfiguration				
Namespace	CORE (voir 3.3)				
TypeDefinition	ServerConfigurationType défini en 7.7.3.				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.7.3 ServerConfigurationType

Ce type définit un *ObjectType* qui représente la configuration d'un *Serveur* prenant en charge la gestion en flux poussé. Ses valeurs sont définies dans le Tableau 40. Il existe toujours précisément une instance dans l'*AddressSpace du Serveur*.

Tableau 40 – Définition de ServerConfigurationType

Attribut	Valeur				
BrowseName	ServerConfigurationType				
Namespace	CORE (voir 3.3)				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**Type Definition**	**Modelling Rule**
Sous-type du *BaseObjectType* défini dans l'IEC 62541-5.					
HasComponent	Objet	CertificateGroups		CertificateGroup FolderType	Obligatoire
HasProperty	Variable	ServerCapabilities	Chaîne[]	PropertyType	Obligatoire
HasProperty	Variable	SupportedPrivateKeyFormats	Chaîne[]	PropertyType	Obligatoire
HasProperty	Variable	MaxTrustListSize	UInt32	PropertyType	Obligatoire
HasProperty	Variable	MulticastDnsEnabled	Booléen	PropertyType	Obligatoire
HasComponent	Méthode	UpdateCertificate	Voir 7.7.4		Obligatoire
HasComponent	Méthode	ApplyChanges	Voir 7.7.5		Facultative
HasComponent	Méthode	CreateSigningRequest	Voir 7.7.6		Obligatoire
HasComponent	Méthode	GetRejectedList	Voir 7.7.7		Obligatoire

L'*Objet CertificateGroups* organise les *Groupes de Certificats* pris en charge par le *Serveur*. Il est décrit en 7.5.17. Les *Serveurs* doivent prendre en charge le *DefaultApplicationGroup* et peuvent prendre en charge le *DefaultHttpsGroup* ou le *DefaultUserTokenGroup*. Les *Serveurs* peuvent prendre en charge des *Groupes de Certificats* supplémentaires en fonction de leurs exigences. Par exemple, il convient qu'un *Serveur* possédant deux interfaces réseau dispose d'une *Liste de Confiance* différente pour chaque interface. La seconde *Liste de Confiance* serait représentée comme un nouvel *Objet CertificateGroupType* organisé par le *Dossier CertificateGroups*.

La *Propriété ServerCapabilities* spécifie les fonctionnalités prises en charge par le *Serveur* et décrites à l'Annexe D. La valeur est identique à celle signalée au *LocalDiscoveryServer* lorsque le *Serveur* appelle le *Service RegisterServer2*.

La *Propriété SupportedPrivateKeyFormats* spécifie les formats de *PrivateKey* pris en charge par le Serveur. Les valeurs possibles incluent "PEM" (voir IETF RFC 5958) ou "PFX" (voir PKCS #12). La matrice est vide si le *Serveur* ne permet pas aux *Clients* externes de mettre à jour la *PrivateKey*.

La *Propriété MaxTrustListSize* spécifie la taille maximale de la *Liste de Confiance*, en octets. 0 indique qu'il n'existe pas de limite. La valeur par défaut est 65 535 octets.

Si *MulticastDnsEnabled* est défini sur TRUE, le *Serveur* s'annonce en utilisant le protocole Multicast DNS. Il peut être modifié en écrivant dans la *Variable*.

La *Méthode GetRejectedList* renvoie la liste des *Certificats* rejetés par le *Serveur*. Elle peut être utilisée pour suivre l'activité ou pour autoriser les administrateurs à déplacer un *Certificat* rejeté vers la *Liste de Confiance*.

La *Méthode UpdateCertificate* est utilisée pour mettre un *Certificat* à jour.

La *Méthode ApplyChanges* est utilisée pour appliquer une quelconque modification de sécurité si le *Serveur* définit le fanion *applyChangesRequired* lorsqu'une autre *Méthode* est appelée. Il convient que les *Serveurs* réduisent le plus possible l'impact de l'application de la nouvelle configuration; toutefois, cela pourrait exiger la fermeture et la réouverture de toutes les *Sessions* existantes par les *Clients*.

La *Méthode CreateSigningRequest* demande au *Serveur* de créer une *Demande de Certificat* à codage *PKCS #10* signée avec la clé privée du *Serveur*.

7.7.4 UpdateCertificate

La *Méthode UpdateCertificate* est utilisée pour mettre à jour le *Certificat* d'un *Serveur*.

Il existe trois cas d'utilisation pour cette *Méthode*:

- le nouveau *Certificat* a été créé sur la base d'une demande de signature créée avec la *Méthode CreateSigningRequest* définie en 7.7.6. Dans ce cas, aucune *privateKey* n'est fournie;
- une nouvelle *privateKey* et un nouveau *Certificat* ont été créés en dehors du *Serveur*; tous deux sont mis à jour avec la présente *Méthode*;
- un nouveau *Certificat* a été créé et signé avec les informations de l'ancien *Certificat*. Dans ce cas, aucune *privateKey* n'est fournie.

Le *Serveur* doit réaliser l'ensemble des contrôles d'intégrité habituels sur le *Certificat* et sur tous les *Certificats* d'émetteur. Si des erreurs se produisent, le code *Bad_SecurityChecksFailed* est renvoyé.

Le *Serveur* doit signaler une erreur si la clé publique ne correspond pas au *Certificat* existant et si la privateKey n'a pas été fournie.

Si le *Serveur* renvoie *applyChangesRequired*=FALSE, il indique qu'il est capable de satisfaire aux exigences spécifiées pour la *Méthode ApplyChanges*.

Cette *Méthode* exige l'utilisation d'un canal chiffré et la fourniture, par le Client, d'authentifiants avec droits administratifs sur le Serveur.

Signature

```
UpdateCertificate(
        [in] NodeId certificateGroupId
        [in] NodeId certificateTypeId
        [in] ByteString certificate
        [in] ByteString[] issuerCertificates
        [in] String privateKeyFormat
        [in] ByteString privateKey
          [out] Boolean applyChangesRequired
    );
```

Argument	Description
certificateGroupId	NodeId de l'*Objet CertificateGroup* affecté par la mise à jour. S'il est nul, le *DefaultApplicationGroup* est utilisé.
certificateTypeId	Type de *Certificat* en cours de mise à jour. L'ensemble des types admis est spécifié par la *Propriété CertificateTypes* appartenant au *Groupe de Certificats*.
certificate	*Certificat* à codage DER qui remplace le Certificat existant.
issuerCertificates	*Certificats* d'émetteur nécessaires pour vérifier la signature sur le nouveau *Certificat*.
privateKeyFormat	Format de la *Clé Privée* (PEM ou PFX). Si la *privateKey* n'est pas spécifiée, le *privateKeyFormat* est nul ou vide.
privateKey	*Clé Privée* codée au *privateKeyFormat*.
applyChangesRequired	Indique que la *Méthode ApplyChanges* doit être appelée avant d'utiliser le nouveau *Certificat*.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_InvalidArgument	Le certificateTypeId ou le certificateGroupId n'est pas valide.
Bad_CertificateInvalid	Le *Certificat* n'est pas valide ou le format n'est pas pris en charge.
Bad_NotSupported	La *PrivateKey* n'est pas valide ou le format n'est pas pris en charge.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.
Bad_SecurityChecksFailed	Une défaillance est survenue lors du contrôle d'intégrité du *Certificat*.

Le Tableau 41 spécifie la représentation de l'*AddressSpace* pour la *Méthode UpdateCertificate*.

Tableau 41 – Définition de l'AddressSpace pour la Méthode UpdateCertificate

Attribut	Valeur				
BrowseName	UpdateCertificate				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.7.5 ApplyChanges

La *Méthode ApplyChanges* est utilisée pour indiquer au *Serveur* d'appliquer une quelconque modification de sécurité.

Il convient d'appeler cette *Méthode* uniquement si un appel précédent d'une *Méthode* qui a modifié la configuration renvoie applyChangesRequired=true (voir 7.7.4).

Si le *Certificat du Serveur* a été modifié, les *Canaux Sécurisés* utilisant l'ancien *Certificat* sont finalement interrompus. La seule latitude dont dispose le *Serveur* concerne les délais. Dans le meilleur des cas, le *Serveur* peut fermer les *TransportConnections* pour les *Points d'extrémité* affectés et laisser les *Abonnements* inchangés. Il convient que l'opération s'apparente à une interruption réseau du point de vue du *Client*. Il convient que le *Client* soit préparé à traiter les modifications de *Certificat* pendant sa logique de reconnexion. Dans le cas le plus défavorable, un arrêt complet qui affecte l'ensemble des *Clients* connectés est nécessaire. Dans ce dernier cas, le *Serveur* doit faire part de son intention d'interrompre les connexions en définissant les *Propriétés SecondsTillShutdown* et *ShutdownReason* dans la *Variable ServerStatus*.

Si le *Canal Sécurisé* actuellement utilisé pour appeler cette *Méthode* est affecté par la modification de *Certificat*, le *Serveur* doit alors introduire un délai suffisant pour permettre à l'appelant de recevoir une réponse.

Cette *Méthode* exige l'utilisation d'un canal chiffré et la fourniture, par le *Client*, d'authentifiants avec droits administratifs sur le *Serveur*.

Signature

```
ApplyChanges();
```

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 42 spécifie la représentation de l'*AddressSpace* pour la *Méthode ApplyChanges*.

Tableau 42 – Définition de l'AddressSpace pour la Méthode ApplyChanges

Attribut	Valeur				
BrowseName	ApplyChanges				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**

7.7.6 CreateSigningRequest

La *Méthode CreateSigningRequest* demande au *Serveur* de créer une *Demande de Certificat* à codage DER *PKCS #10* signée avec la clé privée du *Serveur*. Cette demande peut ensuite être utilisée pour demander un *Certificat* auprès d'une CA qui attend des demandes dans ce format.

Cette *Méthode* exige l'utilisation d'un canal chiffré et la fourniture, par le *Client*, d'authentifiants avec droits administratifs sur le *Serveur*.

Signature

```
CreateSigningRequest(
    [in] NodeId certificateGroupId,
    [in] NodeId certificateTypeId,
    [in] String subjectName,
    [in] Boolean regeneratePrivateKey,
    [in] ByteString nonce,
    [out]    ByteString certificateRequest
    );
```

Argument	Description
certificateGroupId	NodeId de l'*Objet CertificateGroup* affecté par la demande. S'il est nul, le *DefaultApplicationGroup* est utilisé.
certificateTypeId	Type de *Certificat* en cours de demande. L'ensemble des types admis est spécifié par la *Propriété CertificateTypes* appartenant au *Groupe de Certificats*.
subjectName	Nom de sujet à utiliser dans la *Demande de Certificat*. S'il n'est pas spécifié, le *SubjectName* de l'actuel *Certificat* est utilisé. Le format du *subjectName* est défini en 7.6.4.
regeneratePrivateKey	Si l'argument est réglé sur TRUE, le *Serveur* doit créer une nouvelle *Clé Privée* qu'il stocke jusqu'à ce que le *Certificat* signé correspondant soit chargé avec la *Méthode UpdateCertificate*. Les *Clés Privées* précédemment créées peuvent être éliminées si la *Méthode UpdateCertificate* n'a pas été appelée avant de rappeler cette méthode. S'il est réglé sur FALSE, le *Serveur* utilise sa *Clé Privée* existante.
nonce	Entropie supplémentaire que l'appelant doit fournir si regeneratePrivateKey est réglé sur TRUE. Elle doit être d'au moins 32 octets.
certificateRequest	*Demande de Certificat* à codage DER PKCS #10. Si la *CertificateRequest* porte sur un *Certificat d'ApplicationInstance*, elle doit inclure tous les champs exigés par l'IEC 62541-6, comme le *subjectAltName*.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_InvalidArgument	Le *certificateTypeId*, le *certificateGroupId* ou le *subjectName* n'est pas valide.
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 43 spécifie la représentation de l'*AddressSpace* pour la *Méthode CreateSigningRequest*.

Tableau 43 – Définition de l'AddressSpace pour la Méthode CreateSigningRequest

Attribut	Valeur				
BrowseName	CreateSigningRequest				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.7.7 GetRejectedList

La *Méthode GetRejectedList* renvoie la liste des *Certificats* rejetés par le *Serveur*.

Aucune règle n'est définie concernant la manière dont le *Serveur* met cette liste à jour ou la durée de conservation d'un *Certificat* dans la liste. Il est recommandé d'ajouter chaque *Certificat* valide, mais non sécurisé, à la liste des certificats rejetés tant que le stockage est disponible. Il convient que les *Serveurs* ignorent les entrées plus anciennes de la liste renvoyée si la taille de message maximale n'est pas suffisamment importante pour permettre le renvoi de la liste entière.

Cette *Méthode* exige l'utilisation d'un canal chiffré et la fourniture, par le *Client*, d'authentifiants avec droits administratifs sur le *Serveur*.

Signature

```
GetRejectedList(
     [out] ByteString[] certificates
     );
```

Argument	Description
certificates	Forme à codage DER des Certificats rejetés par le Serveur.

Codes de résultat de la méthode (définis dans le Service d'Appel)

Code de résultat	Description
Bad_UserAccessDenied	L'utilisateur actuel ne dispose pas des droits exigés.

Le Tableau 44 spécifie la représentation de l'*AddressSpace* pour la *Méthode GetRejectedList*.

Tableau 44 – Définition de l'AddressSpace pour la Méthode GetRejectedList

Attribut	Valeur				
BrowseName	GetRejectedList				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	OutputArguments	Argument	PropertyType	Obligatoire

7.7.8 CertificateUpdatedAuditEventType

Cet événement est généré lorsque le *Certificat* d'*Application* est modifié.

Il est le résultat de l'exécution réussie ou de l'échec d'une *Méthode UpdateCertificate*.

Sa représentation dans l'*AddressSpace* est définie de manière formelle dans le Tableau 45.

Tableau 45 – Définition de CertificateUpdatedAuditEventType

Attribut	Valeur				
BrowseName	CertificateUpdatedAuditEventType				
Namespace	CORE (voir 3.3)				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AuditUpdateMethodEventType* défini dans l'IEC 62541-5.					
HasProperty	Variable	CertificateGroup	NodeId	PropertyType	Obligatoire
HasProperty	Variable	CertificateType	NodeId	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateMethodEventType*. Leur sémantique est définie dans l'IEC 62541-5.

La *Propriété CertificateGroup* spécifie le *Groupe de Certificats* affecté par la mise à jour.

La *Propriété CertificateType* spécifie le type de *Certificat* mis à jour.

8 Gestion des KeyCredentials

8.1 Vue d'ensemble

Les fonctions de gestion des *KeyCredentials* permettent de gérer et de distribuer les *KeyCredentials* que les *Applications OPC UA* utilisent pour accéder aux *Services d'Autorisation* et/ou aux *Courtiers*. Une application qui fournit les fonctions de gestion des *KeyCredentials* est appelée *KeyCredentialService*; elle est généralement combinée au GDS au sein d'une même application.

Il existe deux principaux modèles de gestion des *KeyCredentials*: la gestion en flux tiré et la gestion en flux poussé. Avec la gestion en flux tiré, l'application agit comme un *Client* et utilise les *Méthodes* sur le *KeyCredentialService* pour demander et mettre à jour les *KeyCredentials*. L'application est chargée de garantir que les *KeyCredentials* sont à jour. Avec la gestion en flux poussé, l'application agit comme un *Serveur* et expose les *Méthodes* que le *KeyCredentialService* peut appeler pour mettre à jour les *KeyCredentials* selon les exigences.

Un *KeyCredentialService* peut directement gérer les *KeyCredentials* qu'il fournit ou il peut agir comme un intermédiaire entre le *Client* et un système ne prenant pas en charge OPC UA, tel qu'Azure AD[1] ou LDAP.

Noter que les *KeyCredentials* sont des secrets directement transmis aux *Services d'Autorisation* et/ou aux *Courtiers*; il ne s'agit pas de *Certificats* disposant de clés privées. La distribution des *Certificats* est gérée par le modèle de gestion des *Certificats* décrit à l'Article 7. Par exemple, les *Services d'Autorisation* qui prennent en charge OAuth2 exigent souvent du client qu'il fournisse les paramètres client_id et client_secret avec chaque demande. Les *KeyCredentials* sont les valeurs que l'application doit attribuer à ces paramètres.

8.2 Gestion en flux tiré

La gestion en flux tiré s'effectue en utilisant un *Objet KeyCredentialManagement* (voir 8.4.3). Elle permet aux *Clients* de demander les authentifiants pour les *Services d'Autorisation* ou les *Courtiers* pris en charge par le *KeyCredentialService*. Les interactions entre le *Client* et le *KeyCredentialService* dans le cadre d'une gestion en flux tiré sont représentées à la Figure 16.

[1] Azure AD est l'appellation commerciale d'un produit distribué par Microsoft®. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

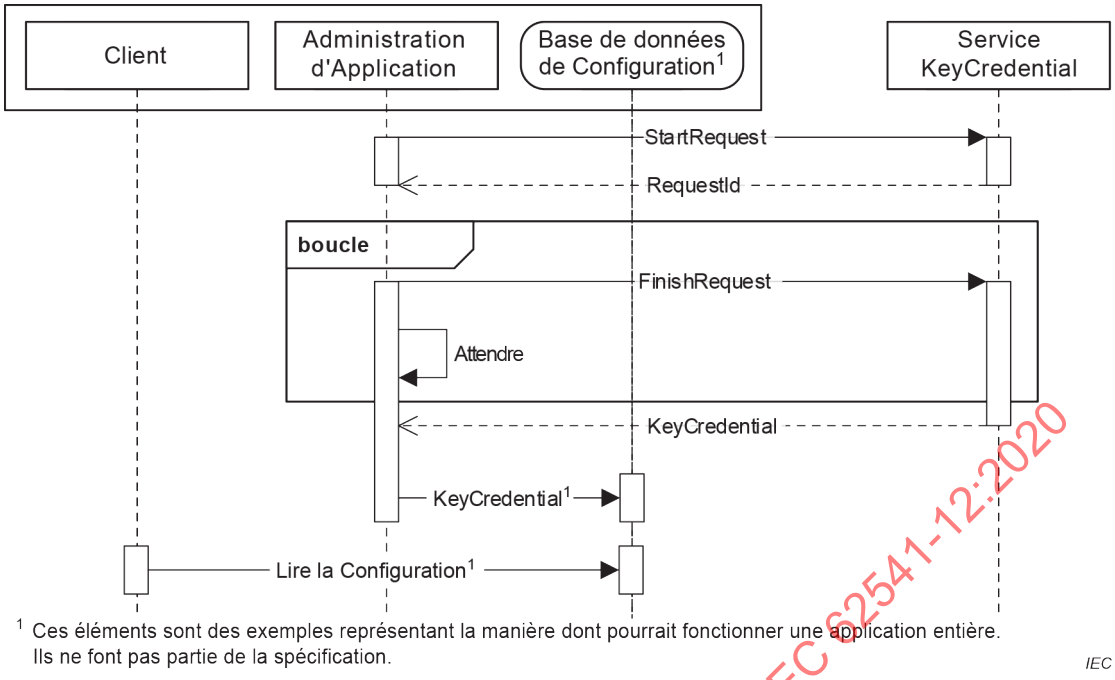


Figure 16 – Modèle de gestion des KeyCredentials en flux tiré

Le composant d'Administration d'Application peut faire partie du *Client* ou d'un service autonome qui comprend la manière dont le *Client* conserve ses informations de configuration dans sa Base de données de Configuration. Les composants d'administration et de base de données sont des exemples qui indiquent la manière dont une application pourrait être conçue et ne constituent pas une exigence.

La demande d'authentifiants est un processus en deux étapes, certains *KeyCredentialServices* exigeant une révision et une approbation des demandes par une personne physique. Les appels de la *Méthode FinishKeyCredentialRequest* peuvent ne pas être réguliers et pourraient être initiés par certains événements, comme un utilisateur qui démarre l'application ou qui interagit avec un élément de l'IU (un bouton, par exemple).

Les *KeyCredentials* peuvent uniquement être demandées pour les *Clients* auxquels le *KeyCredentialService* fait confiance.

Avec le modèle de gestion en flux tiré, la sécurité exige d'utiliser un canal chiffré et les authentifiants d'administrateur du *KeyCredentialService* afin de garantir que seuls les utilisateurs autorisés peuvent demander les *KeyCredentials*.

8.3 Gestion en flux poussé

La gestion en flux poussé s'effectue en utilisant un *Objet KeyCredentialConfiguration* (voir 8.5.3) qui est un composant du *Dossier KeyCredentialManagement*, qui est un composant de l'*Objet ServerConfiguration* d'un *Serveur*. Les interactions entre l'application d'Administration et le *KeyCredentialService* dans le cadre d'une gestion en flux poussé sont représentées à la Figure 17.