

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**OPC Unified Architecture –
Part 4: Services**

**Architecture Unifiée OPC –
Partie 4: Services**

Withdrawn



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2015 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

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

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
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 corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub

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 It also gives information on projects, replaced and withdrawn publications.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 also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

More than 60 000 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.

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: csc@iec.ch.

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.

Catalogue IEC - webstore.iec.ch/catalogue

Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub

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 aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 15 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary

Plus de 60 000 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.

Service Clients - webstore.iec.ch/csc

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

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**OPC Unified Architecture –
Part 4: Services**

**Architecture Unifiée OPC –
Partie 4: Services**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 25.040.40; 35.100

ISBN 978-2-8322-2369-7

Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

CONTENTS

FOREWORD	12
1 Scope	14
2 Normative references	14
3 Terms, definitions and conventions	15
3.1 Terms and definitions	15
3.2 Abbreviations and symbols	15
3.3 Conventions for Service definitions	16
4 Overview	17
4.1 Service Set model	17
4.2 Request/response Service procedures	21
5 Service Sets	21
5.1 General	21
5.2 Service request and response header	21
5.3 Service results	21
5.4 Discovery Service Set	23
5.4.1 Overview	23
5.4.2 FindServers	24
5.4.3 GetEndpoints	25
5.4.4 RegisterServer	28
5.5 SecureChannel Service Set	31
5.5.1 Overview	31
5.5.2 OpenSecureChannel	32
5.5.3 CloseSecureChannel	35
5.6 Session Service Set	36
5.6.1 Overview	36
5.6.2 CreateSession	36
5.6.3 ActivateSession	40
5.6.4 CloseSession	43
5.6.5 Cancel	43
5.7 NodeManagement Service Set	44
5.7.1 Overview	44
5.7.2 AddNodes	44
5.7.3 AddReferences	46
5.7.4 DeleteNodes	47
5.7.5 DeleteReferences	48
5.8 View Service Set	49
5.8.1 Overview	49
5.8.2 Browse	50
5.8.3 BrowseNext	52
5.8.4 TranslateBrowsePathsToNodeIds	54
5.8.5 RegisterNodes	56
5.8.6 UnregisterNodes	57
5.9 Query Service Set	57
5.9.1 Overview	57
5.9.2 Querying Views	58
5.9.3 QueryFirst	58

5.9.4	QueryNext	62
5.10	Attribute Service Set	63
5.10.1	Overview	63
5.10.2	Read	63
5.10.3	HistoryRead	65
5.10.4	Write	67
5.10.5	HistoryUpdate	69
5.11	Method Service Set	70
5.11.1	Overview	70
5.11.2	Call	71
5.12	MonitoredItem Service Set	73
5.12.1	MonitoredItem model	73
5.12.2	CreateMonitoredItems	79
5.12.3	ModifyMonitoredItems	81
5.12.4	SetMonitoringMode	83
5.12.5	SetTriggering	84
5.12.6	DeleteMonitoredItems	85
5.13	Subscription Service Set	86
5.13.1	Subscription model	86
5.13.2	CreateSubscription	93
5.13.3	ModifySubscription	94
5.13.4	SetPublishingMode	95
5.13.5	Publish	96
5.13.6	Republish	98
5.13.7	TransferSubscriptions	98
5.13.8	DeleteSubscriptions	100
6	Service behaviours	101
6.1	Security	101
6.1.1	Overview	101
6.1.2	Obtaining and Installing an Application Instance Certificate	101
6.1.3	Determining if a Certificate is Trusted	102
6.1.4	Creating a SecureChannel	104
6.1.5	Creating a Session	106
6.1.6	Impersonating a User	107
6.2	Software Certificates	107
6.2.1	Overview	107
6.2.2	Obtaining and Installing a Software Certificate	107
6.2.3	Validating a Software Certificate	109
6.3	Auditing	109
6.3.1	Overview	109
6.3.2	General audit logs	109
6.3.3	General audit Events	109
6.3.4	Auditing for Discovery Service Set	110
6.3.5	Auditing for SecureChannel Service Set	110
6.3.6	Auditing for Session Service Set	110
6.3.7	Auditing for NodeManagement Service Set	111
6.3.8	Auditing for Attribute Service Set	111
6.3.9	Auditing for Method Service Set	111
6.3.10	Auditing for View, Query, MonitoredItem and Subscription Service Set	112

6.4	Redundancy.....	112
6.4.1	Redundancy overview.....	112
6.4.2	Server redundancy overview.....	112
6.4.3	Client redundancy.....	116
6.4.4	Network redundancy.....	116
6.5	Re-establishing connections.....	117
7	Common parameter type definitions.....	118
7.1	ApplicationDescription.....	118
7.2	ApplicationInstanceCertificate.....	118
7.3	BrowseResult.....	119
7.4	ContentFilter.....	119
7.4.1	ContentFilter structure.....	119
7.4.2	ContentFilterResult.....	120
7.4.3	FilterOperator.....	121
7.4.4	FilterOperand parameters.....	127
7.5	Counter.....	128
7.6	ContinuationPoint.....	129
7.7	DataValue.....	129
7.7.1	General.....	129
7.7.2	PicoSeconds.....	130
7.7.3	SourceTimestamp.....	130
7.7.4	ServerTimestamp.....	130
7.7.5	StatusCode assigned to a value.....	131
7.8	DiagnosticInfo.....	131
7.9	EndpointDescription.....	132
7.10	ExpandedNodeId.....	133
7.11	ExtensibleParameter.....	133
7.12	Index.....	134
7.13	IntegerId.....	134
7.14	MessageSecurityMode.....	134
7.15	MonitoringParameters.....	134
7.16	MonitoringFilter parameters.....	135
7.16.1	Overview.....	135
7.16.2	DataChangeFilter.....	136
7.16.3	EventFilter.....	136
7.16.4	AggregateFilter.....	139
7.17	MonitoringMode.....	140
7.18	NodeAttributes parameters.....	140
7.18.1	Overview.....	140
7.18.2	ObjectAttributes parameter.....	141
7.18.3	VariableAttributes parameter.....	141
7.18.4	MethodAttributes parameter.....	142
7.18.5	ObjectTypeAttributes parameter.....	142
7.18.6	VariableTypeAttributes parameter.....	142
7.18.7	ReferenceTypeAttributes parameter.....	143
7.18.8	DataTypeAttributes parameter.....	143
7.18.9	ViewAttributes parameter.....	143
7.19	NotificationData parameters.....	144
7.19.1	Overview.....	144

7.19.2	DataChangeNotification parameter	144
7.19.3	EventNotificationList parameter	145
7.19.4	StatusChangeNotification parameter	145
7.20	NotificationMessage	145
7.21	NumericRange	146
7.22	QueryDataSet	147
7.23	ReadValueId	147
7.24	ReferenceDescription	148
7.25	RelativePath	149
7.26	RequestHeader	149
7.27	ResponseHeader	151
7.28	ServiceFault	151
7.29	SessionAuthenticationToken	151
7.30	SignatureData	153
7.31	SignedSoftwareCertificate	153
7.32	SoftwareCertificate	153
7.33	StatusCode	154
7.33.1	General	154
7.33.2	Common StatusCodes	156
7.34	TimestampsToReturn	159
7.35	UserIdentityToken parameters	159
7.35.1	Overview	159
7.35.2	AnonymousIdentityToken	160
7.35.3	UserNameIdentityToken	160
7.35.4	X509IdentityTokens	161
7.35.5	IssuedIdentityToken	161
7.36	UserTokenPolicy	162
7.37	ViewDescription	163
Annex A	(informative) BNF definitions	164
A.1	Overview over BNF	164
A.2	BNF of RelativePath	164
A.3	BNF of NumericRange	165
Annex B	(informative) Content Filter and Query Examples	166
B.1	Simple ContentFilter examples	166
B.1.1	Overview	166
B.1.2	Example 1	166
B.1.3	Example 2	167
B.2	Complex Examples of Query Filters	167
B.2.1	Overview	167
B.2.2	Used type model	168
B.2.3	Example Notes	170
B.2.4	Example 1	171
B.2.5	Example 2	172
B.2.6	Example 3	173
B.2.7	Example 4	175
B.2.8	Example 5	176
B.2.9	Example 6	178
B.2.10	Example 7	179
B.2.11	Example 8	181

B.2.12	Example 9.....	182
Figure 1	– Discovery Service Set	17
Figure 2	– SecureChannel Service Set.....	18
Figure 3	– Session Service Set	18
Figure 4	– NodeManagement Service Set	18
Figure 5	– View Service Set.....	19
Figure 6	– Attribute Service Set	19
Figure 7	– Method Service Set.....	20
Figure 8	– MonitoredItem and Subscription Service Sets	20
Figure 9	– Discovery process.....	23
Figure 10	– Using a Gateway Server.....	27
Figure 11	– The Registration Process – Manually Launched Servers.....	28
Figure 12	– The Registration Process – Automatically Launched Servers.....	29
Figure 13	– SecureChannel and Session Services	32
Figure 14	– Multiplexing Users on a Session.....	38
Figure 15	– MonitoredItem Model.....	74
Figure 16	– Typical delay in change detection.....	75
Figure 17	– Queue overflow handling	77
Figure 18	– Triggering Model	78
Figure 19	– Obtaining and Installing an Application Instance Certificate.....	102
Figure 20	– Determining if a Application Instance Certificate is Trusted	104
Figure 21	– Establishing a SecureChannel.....	105
Figure 22	– Establishing a Session	106
Figure 23	– Impersonating a User	107
Figure 24	– Obtaining and Installing a Software Certificate	108
Figure 25	– Transparent Redundancy setup.....	113
Figure 26	– Non-Transparent Redundancy setup	113
Figure 27	– Server proxy for redundancy	116
Figure 28	– Reconnect Sequence	117
Figure 29	– Logical layers of a Server.....	152
Figure 30	– Obtaining a SessionAuthenticationToken	152
Figure B.1	– Filter Logic Tree Example	166
Figure B.2	– Filter Logic Tree Example	167
Figure B.3	– Example Type Nodes	169
Figure B.4	– Example Instance Nodes	170
Figure B.5	– Example 1 Filter.....	171
Figure B.6	– Example 2 Filter Logic Tree	172
Figure B.7	– Example 3 Filter Logic Tree	174
Figure B.8	– Example 4 Filter Logic Tree	176
Figure B.9	– Example 5 Filter Logic Tree	177
Figure B.10	– Example 6 Filter Logic Tree	178
Figure B.11	– Example 7 Filter Logic Tree	180

Figure B.12 – Example 8 Filter Logic Tree	181
Figure B.13 – Example 9 Filter Logic Tree	183

Table 1 – Service Definition Table	16
Table 2 – Parameter Types defined in IEC 62541-3	17
Table 3 – FindServers Service Parameters	25
Table 4 – GetEndpoints Service Parameters	27
Table 5 – RegisterServer Service Parameters	30
Table 6 – RegisterServer Service Result Codes	30
Table 7 – OpenSecureChannel Service Parameters	34
Table 8 – OpenSecureChannel Service Result Codes	35
Table 9 – CloseSecureChannel Service Parameters	35
Table 10 – CloseSecureChannel Service Result Codes	35
Table 11 – CreateSession Service Parameters	38
Table 12 – CreateSession Service Result Codes	40
Table 13 – ActivateSession Service Parameters	42
Table 14 – ActivateSession Service Result Codes	43
Table 15 – CloseSession Service Parameters	43
Table 16 – CloseSession Service Result Codes	43
Table 17 – Cancel Service Parameters	44
Table 18 – AddNodes Service Parameters	45
Table 19 – AddNodes Service Result Codes	45
Table 20 – AddNodes Operation Level Result Codes	46
Table 21 – AddReferences Service Parameters	46
Table 22 – AddReferences Service Result Codes	47
Table 23 – AddReferences Operation Level Result Codes	47
Table 24 – DeleteNodes Service Parameters	48
Table 25 – DeleteNodes Service Result Codes	48
Table 26 – DeleteNodes Operation Level Result Codes	48
Table 27 – DeleteReferences Service Parameters	49
Table 28 – DeleteReferences Service Result Codes	49
Table 29 – DeleteReferences Operation Level Result Codes	49
Table 30 – Browse Service Parameters	51
Table 31 – Browse Service Result Codes	52
Table 32 – Browse Operation Level Result Codes	52
Table 33 – BrowseNext Service Parameters	53
Table 34 – BrowseNext Service Result Codes	53
Table 35 – BrowseNext Operation Level Result Codes	54
Table 36 – TranslateBrowsePathsToNodeIds Service Parameters	55
Table 37 – TranslateBrowsePathsToNodeIds Service Result Codes	55
Table 38 – TranslateBrowsePathsToNodeIds Operation Level Result Codes	56
Table 39 – RegisterNodes Service Parameters	56
Table 40 – RegisterNodes Service Result Codes	57

Table 41 – UnregisterNodes Service Parameters	57
Table 42 – UnregisterNodes Service Result Codes	57
Table 43 – QueryFirst Request Parameters	60
Table 44 – QueryFirst Response Parameters	61
Table 45 – QueryFirst Service Result Codes	62
Table 46 – QueryFirst Operation Level Result Codes	62
Table 47 – QueryNext Service Parameters	63
Table 48 – QueryNext Service Result Codes	63
Table 49 – Read Service Parameters	64
Table 50 – Read Service Result Codes	64
Table 51 – Read Operation Level Result Codes	65
Table 52 – HistoryRead Service Parameters	66
Table 53 – HistoryRead Service Result Codes	67
Table 54 – HistoryRead Operation Level Result Codes	67
Table 55 – Write Service Parameters	68
Table 56 – Write Service Result Codes	69
Table 57 – Write Operation Level Result Codes	69
Table 58 – HistoryUpdate Service Parameters	70
Table 59 – HistoryUpdate Service Result Codes	70
Table 60 – HistoryUpdate Operation Level Result Codes	70
Table 61 – Call Service Parameters	72
Table 62 – Call Service Result Codes	72
Table 63 – Call Operation Level Result Codes	73
Table 64 – CreateMonitoredItems Service Parameters	80
Table 65 – CreateMonitoredItems Service Result Codes	80
Table 66 – CreateMonitoredItems Operation Level Result Codes	81
Table 67 – ModifyMonitoredItems Service Parameters	82
Table 68 – ModifyMonitoredItems Service Result Codes	82
Table 69 – ModifyMonitoredItems Operation Level Result Codes	83
Table 70 – SetMonitoringMode Service Parameters	83
Table 71 – SetMonitoringMode Service Result Codes	83
Table 72 – SetMonitoringMode Operation Level Result Codes	84
Table 73 – SetTriggering Service Parameters	84
Table 74 – SetTriggering Service Result Codes	85
Table 75 – SetTriggering Operation Level Result Codes	85
Table 76 – DeleteMonitoredItems Service Parameters	85
Table 77 – DeleteMonitoredItems Service Result Codes	86
Table 78 – DeleteMonitoredItems Operation Level Result Codes	86
Table 79 – Subscription States	88
Table 80 – Subscription State Table	89
Table 81 – State variables and parameters	91
Table 82 – Functions	92
Table 83 – CreateSubscription Service Parameters	93

Table 84 – CreateSubscription Service Result Codes	94
Table 85 – ModifySubscription Service Parameters	94
Table 86 – ModifySubscription Service Result Codes	95
Table 87 – SetPublishingMode Service Parameters	95
Table 88 – SetPublishingMode Service Result Codes	96
Table 89 – SetPublishingMode Operation Level Result Codes	96
Table 90 – Publish Service Parameters	97
Table 91 – Publish Service Result Codes	97
Table 92 – Publish Operation Level Result Codes	98
Table 93 – Republish Service Parameters	98
Table 94 – Republish Service Result Codes	98
Table 95 – TransferSubscriptions Service Parameters	99
Table 96 – TransferSubscriptions Service Result Codes	99
Table 97 – TransferSubscriptions Operation Level Result Codes	100
Table 98 – DeleteSubscriptions Service Parameters	100
Table 99 – DeleteSubscriptions Service Result Codes	100
Table 100 – DeleteSubscriptions Operation Level Result Codes	101
Table 101 – Certificate Validation Steps	103
Table 102 – Redundancy failover actions	114
Table 103 – ApplicationDescription	118
Table 104 – ApplicationInstanceCertificate	119
Table 105 – BrowseResult	119
Table 106 – ContentFilter Structure	120
Table 107 – ContentFilterResult Structure	120
Table 108 – ContentFilterResult Result Codes	120
Table 109 – ContentFilterResult Operand Result Codes	121
Table 110 – Basic FilterOperator Definition	121
Table 111 – Complex FilterOperator Definition	123
Table 112 – Wildcard characters	124
Table 113 – Conversion Rules	125
Table 114 – Data Precedence Rules	126
Table 115 – Logical AND Truth Table	126
Table 116 – Logical OR Truth Table	127
Table 117 – FilterOperand parameter Typelds	127
Table 118 – ElementOperand	127
Table 119 – LiteralOperand	127
Table 120 – AttributeOperand	128
Table 121 – SimpleAttributeOperand	128
Table 122 – DataValue	129
Table 123 – DiagnosticInfo	132
Table 124 – EndpointDescription	133
Table 125 – ExpandedNodeId	133
Table 126 – ExtensibleParameter Base Type	134

Table 127 – MessageSecurityMode Values	134
Table 128 – MonitoringParameters	135
Table 129 – MonitoringFilter parameterTypelds	136
Table 130 – DataChangeFilter	136
Table 131 – EventFilter structure	138
Table 132 – EventFilterResult structure	138
Table 133 – EventFilterResult Result Codes	138
Table 134 – AggregateFilter structure	139
Table 135 – AggregateFilterResult structure	140
Table 136 – MonitoringMode Values	140
Table 137 – NodeAttributes parameterTypelds	140
Table 138 – Bit mask for specified Attributes	141
Table 139 – ObjectAttributes	141
Table 140 – VariableAttributes	142
Table 141 – MethodAttributes	142
Table 142 – ObjectTypeAttributes	142
Table 143 – VariableTypeAttributes	143
Table 144 – ReferenceTypeAttributes	143
Table 145 – DataTypeAttributes	143
Table 146 – ViewAttributes	144
Table 147 – NotificationData parameterTypelds	144
Table 148 – DataChangeNotification	145
Table 149 – EventNotificationList	145
Table 150 – StatusChangeNotification	145
Table 151 – NotificationMessage	146
Table 152 – NumericRange	147
Table 153 – QueryDataSet	147
Table 154 – ReadValueId	148
Table 155 – ReferenceDescription	148
Table 156 – RelativePath	149
Table 157 – RequestHeader	150
Table 158 – ResponseHeader	151
Table 159 – ServiceFault	151
Table 160 – SignatureData	153
Table 161 – SignedSoftwareCertificate	153
Table 162 – SoftwareCertificate	154
Table 163 – StatusCode Bit Assignments	155
Table 164 – DataValue InfoBits	156
Table 165 – Common Service Result Codes	157
Table 166 – Common Operation Level Result Codes	158
Table 167 – TimestampsToReturn Values	159
Table 168 – UserIdentityToken parameterTypelds	159
Table 169 – UserIdentityToken Encrypted Token Format	160

Table 170 – AnonymousIdentityToken	160
Table 171 – UserNameIdentityToken	161
Table 172 – EncryptionAlgorithm selection.....	161
Table 173 – X509 Identity Token.....	161
Table 174 – IssuedIdentityToken	162
Table 175 – UserTokenPolicy	162
Table 176 – ViewDescription.....	163
Table A.1 – RelativePath	164
Table A.2 – <i>RelativePath</i> Examples	165
Table B.1 – ContentFilter Example.....	167
Table B.2 – ContentFilter Example.....	167
Table B.3 – Example 1 NodeTypeDescription	171
Table B.4 – Example 1 ContentFilter.....	171
Table B.5 – Example 1 QueryDataSets	172
Table B.6 – Example 2 NodeTypeDescription	172
Table B.7 – Example 2 ContentFilter.....	173
Table B.8 – Example 2 QueryDataSets	173
Table B.9 – Example 3 – NodeTypeDescription	173
Table B.10 – Example 3 ContentFilter.....	175
Table B.11 – Example 3 QueryDataSets	175
Table B.12 – Example 4 NodeTypeDescription.....	176
Table B.13 – Example 4 ContentFilter.....	176
Table B.14 – Example 4 QueryDataSets	176
Table B.15 – Example 5 NodeTypeDescription.....	177
Table B.16 – Example 5 ContentFilter.....	177
Table B.17 – Example 5 QueryDataSets	177
Table B.18 – Example 6 NodeTypeDescription.....	178
Table B.19 – Example 6 ContentFilter.....	178
Table B.20 – Example 6 QueryDataSets	179
Table B.21 – Example 6 QueryDataSets without Additional Information	179
Table B.22 – Example 7 NodeTypeDescription.....	180
Table B.23 – Example 7 ContentFilter.....	180
Table B.24 – Example 7 QueryDataSets	181
Table B.25 – Example 8 NodeTypeDescription.....	181
Table B.26 – Example 8 ContentFilter.....	182
Table B.27 – Example 8 QueryDataSets	182
Table B.28 – Example 9 NodeTypeDescription.....	182
Table B.29 – Example 9 ContentFilter.....	183
Table B.30 – Example 9 QueryDataSets	183

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC Unified Architecture –

Part 4: 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 62541-4 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2011. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- a) Update for 6.4 Redundancy.
Added non-transparent redundancy option HotAndMirrored and reworked most of the redundancy description.
- b) Clarifications for Publish and Reconnect scenarios.
Reworked different parts of the specification to make sure no data is lost during short communication interruptions and clients can always detect for how long they

information during connection interruption. Added new clause 6.5 Re-establishing connections that describes the exact reconnect sequence for clients loosing connection to a server. Changed the minimum requirement for the retransmission queue of sent NotificationMessages from one keep-alive interval to minimum two times the minimum number of Publish requests per Session. Added clarification in which data value the overflow bit is set depending on the discard oldest setting. Changed discard handling for discardOldest is FALSE. The new value is replacing the last value put into the queue for FALSE. Added exception that the overflow bit is not set if the queue size one.

- c) Handling of MonitoredItem changes in short network interruption scenarios.
Added new method GetMonitoredItems in Part 5. This method can be used to get the list of monitored items in a subscription if CreateMonitoredItems failed due to a network interruption and the client does not know if the creation succeeded in the server.
- d) Update for 6.1.3 Determining if a Certificate is Trusted
Revised rules for certificate validation.
- e) Revised definition of parameters semaphoreFile and isOnline in Service RegisterServer
- f) Services ModifySubscription and ModifyMonitoredItems
Clarified that changes are applied directly and will take effect as soon as practical but not later than twice the new time interval.
- g) There is a long list of minor changes to eliminate ambiguity.

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

CDV	Report on voting
65E/375/CDV	65E/403/RVC

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

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

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 publication will remain unchanged until the stability date indicated on the IEC web site under "<http://webstore.iec.ch>" in the data related to the specific publication. At this date, the publication will be

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

IMPORTANT – The “colour inside” logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this publication using a colour printer.

OPC Unified Architecture –

Part 4: Services

1 Scope

This part of IEC 62541 defines the OPC Unified Architecture (OPC UA) *Services*. The *Services* described are the collection of abstract Remote Procedure Calls (RPC) that are implemented by OPC UA *Servers* and called by OPC UA *Clients*. All interactions between OPC UA *Clients* and *Servers* occur via these *Services*. The defined *Services* are considered abstract because no particular RPC mechanism for implementation is defined in this part. IEC 62541-6 specifies one or more concrete mappings supported for implementation. For example, one mapping in IEC 62541-6 is to XML Web Services. In that case the *Services* described in this part appear as the Web service methods in the WSDL contract.

Not all OPC UA *Servers* will need to implement all of the defined *Services*. IEC 62541-7 defines the *Profiles* that dictate which *Services* need to be implemented in order to be compliant with a particular *Profile*.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 62541-5, *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 62541-8, *OPC unified architecture – Part 8: Data Access*

IEC 62541-11, *OPC Unified Architecture – Part 11: Historical Access*

IEC 62541-13 *OPC Unified Architecture – Part 13: Aggregates*

3 Terms, definitions 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, and IEC 62541-3, as well as the following apply.

3.1.1

Deadband

permitted range for value changes that will not trigger a data change *Notification*

Note 1 to entry: *Deadband* can be applied as a filter when subscribing to *Variables* and is used to keep noisy signals from updating the *Client* unnecessarily. This standard defines *AbsoluteDeadband* as a common filter. IEC 62541-8 defines an additional *Deadband* filter.

3.1.2

Endpoint

physical address available on a network that allows *Clients* to access one or more *Services* provided by a *Server*

Note 1 to entry: Each *Server* may have multiple *Endpoints*. The address of an *Endpoint* shall include a *HostName*.

3.1.3

Gateway Server

Server that acts as an intermediary for one or more *Servers*

Note 1 to entry: *Gateway Servers* may be deployed to limit external access, provide protocol conversion or to provide features that the underlying *Servers* do not support.

3.1.4

Hostname

unique identifier for a machine on a network

Note 1 to entry: This identifier shall be unique within a local network; however, it may also be globally unique. The identifier can be an IP address.

3.1.5

Security Token

identifier for a cryptographic key set

Note 1 to entry: All *Security Tokens* belong to a security context which is in case of OPC UA the *Secure Channel*.

3.1.6

SoftwareCertificate

digital certificate for a software product that can be installed on several hosts to describe the capabilities of the software product

Note 1 to entry: Different installations of one software product could have the same software certificate. Software certificates are not relevant for security. They are used to identify a software product and its supported features. *SoftwareCertificates* are described in 6.2.

3.2 Abbreviations and symbols

API	Application Programming Interface
BNF	Backus-Naur Form
CA	Certificate Authority
CRL	Certificate Revocation List
CTL	Certificate Trust List

DA Data Access
 NAT Network Address Translation
 UA Unified Architecture
 URI Uniform Resource Identifier
 URL Uniform Resource Locator

3.3 Conventions for Service definitions

OPC UA *Services* contain parameters that are conveyed between the *Client* and the *Server*. The OPC UA *Service* specifications use tables to describe *Service* parameters, as shown in Table 1. Parameters are organised in this table into request parameters and response parameters.

Table 1 – Service Definition Table

Name	Type	Description
Request		Defines the request parameters of the <i>Service</i>
Simple Parameter Name		Description of this parameter
Constructed Parameter Name		Description of the constructed parameter
Component Parameter Name		Description of the component parameter
Response		Defines the response parameters of the <i>Service</i>

The Name, Type and Description columns contain the name, data type and description of each parameter. All parameters are mandatory, although some may be unused under certain circumstances. The Description column specifies the value to be supplied when a parameter is unused.

Two types of parameters are defined in these tables, simple and constructed. Simple parameters have a simple data type, such as *Boolean* or *String*.

Constructed parameters are composed of two or more component parameters, which can be simple or constructed. Component parameter names are indented below the constructed parameter name.

The data types used in these tables may be base types, common types to multiple *Services* or *Service*-specific types. Base data types are defined in IEC 62541-3. The base types used in *Services* are listed in Table 2. Data types that are common to multiple *Services* are defined in Clause 7. Data types that are *Service*-specific are defined in the parameter table of the *Service*.

Table 2 – Parameter Types defined in IEC 62541-3

Parameter Type
BaseDataType
Boolean
ByteString
Double
Duration
Guid
Int32
LocaleId
NodeId
QualifiedName
String
UInt16
UInt32
UInteger
UtcTime
XmlElement

The parameters of the Request and Indication service primitives are represented in Table 1 as Request parameters. Likewise, the parameters of the Response and Confirmation service primitives are represented in Table 1 as Response parameters. All request and response parameters are conveyed between the sender and receiver without change. Therefore, separate columns for request, indication, response and confirmation parameter values are not needed and have been intentionally omitted to improve readability.

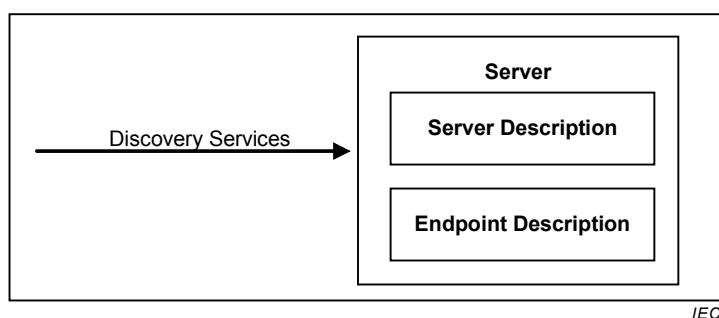
4 Overview

4.1 Service Set model

This clause specifies the OPC UA *Services*. The OPC UA *Service* definitions are abstract descriptions and do not represent a specification for implementation. The mapping between the abstract descriptions and the *Communication Stack* derived from these *Services* are defined in IEC 62541-6. In the case of an implementation as web services, the OPC UA *Services* correspond to the web service and an OPC UA *Service* corresponds to an operation of the web service.

These *Services* are organised into *Service Sets*. Each *Service Set* defines a set of related *Services*. The organisation in *Service Sets* is a logical grouping used in this standard and is not used in the implementation.

The *Discovery Service Set*, illustrated in Figure 1, defines *Services* that allow a *Client* to discover the *Endpoints* implemented by a *Server* and to read the security configuration for each of those *Endpoints*.



IEC

Figure 1 – Discovery Service Set

The *SecureChannel Service Set*, illustrated in Figure 2, defines *Services* that allow a *Client* to establish a communication channel to ensure the *Confidentiality* and *Integrity of Messages* exchanged with the *Server*.

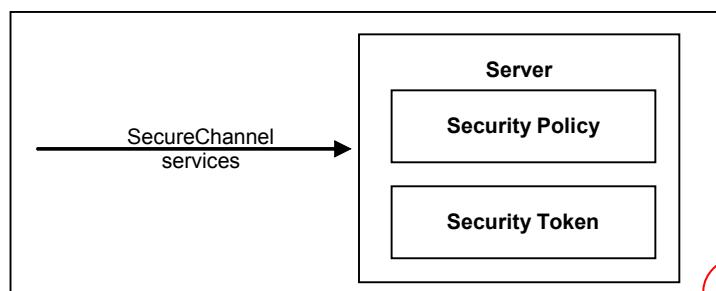


Figure 2 – SecureChannel Service Set

The *Session Service Set*, illustrated in Figure 3, defines *Services* that allow the *Client* to authenticate the user on whose behalf it is acting and to manage *Sessions*.

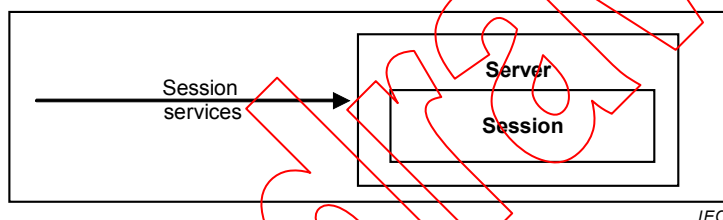


Figure 3 – Session Service Set

The *NodeManagement Service Set*, illustrated in Figure 4, defines *Services* that allow the *Client* to add, modify and delete *Nodes* in the *AddressSpace*.

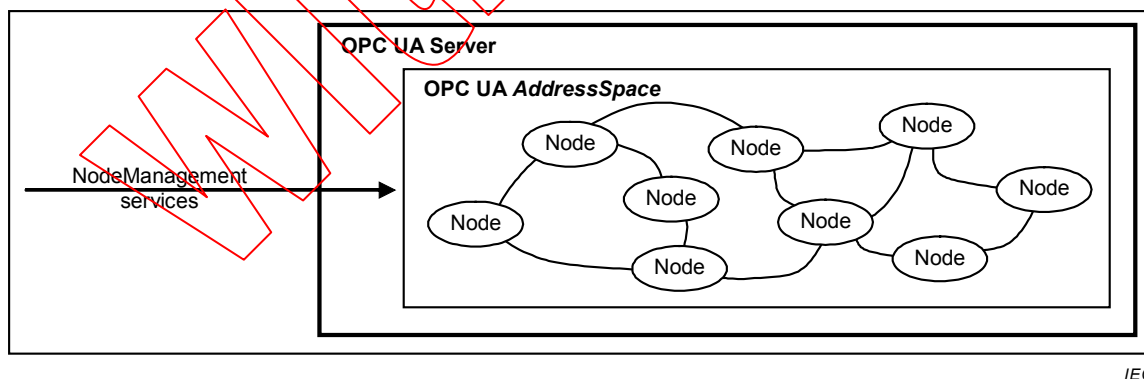


Figure 4 – NodeManagement Service Set

The *View Service Set*, illustrated in Figure 5, defines *Services* that allow *Clients* to browse through the *AddressSpace* or subsets of the *AddressSpace* called *Views*. The *Query Service Set* allows *Clients* to get a subset of data from the *AddressSpace* or the *View*.

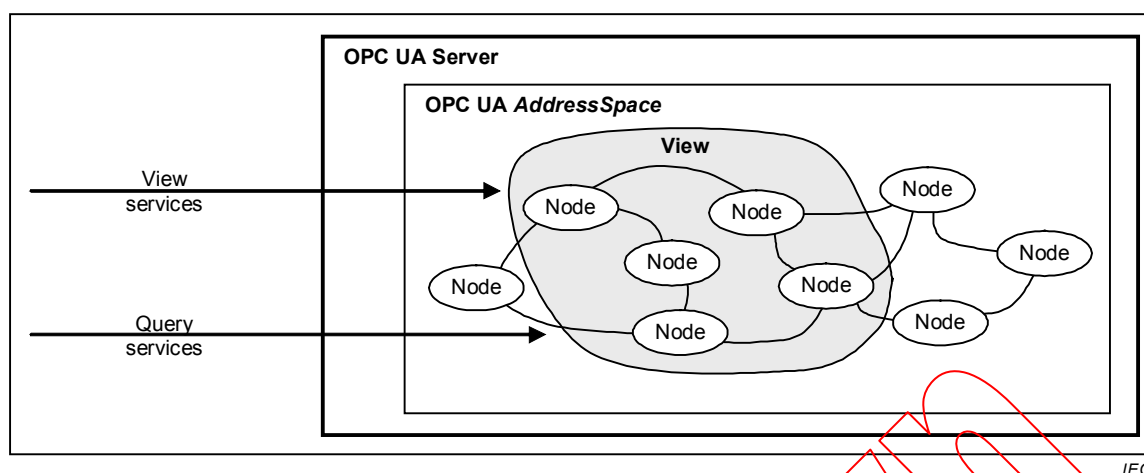


Figure 5 – View Service Set

The *Attribute Service Set* is illustrated in Figure 6. It defines *Services* that allow *Clients* to read and write *Attributes* of *Nodes*, including their historical values. Since the value of a *Variable* is modelled as an *Attribute*, these *Services* allow *Clients* to read and write the values of *Variables*.

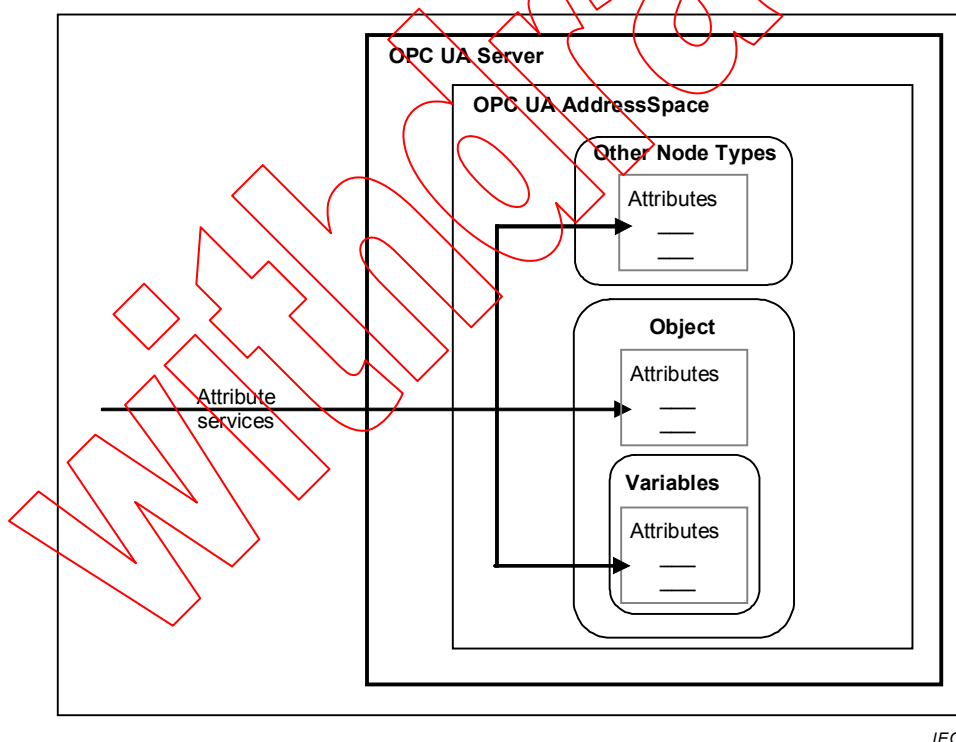


Figure 6 – Attribute Service Set

The *Method Service Set* is illustrated in Figure 7. It defines *Services* that allow *Clients* to call methods. Methods run to completion when called. They may be called with method-specific input parameters and may return method-specific output parameters.

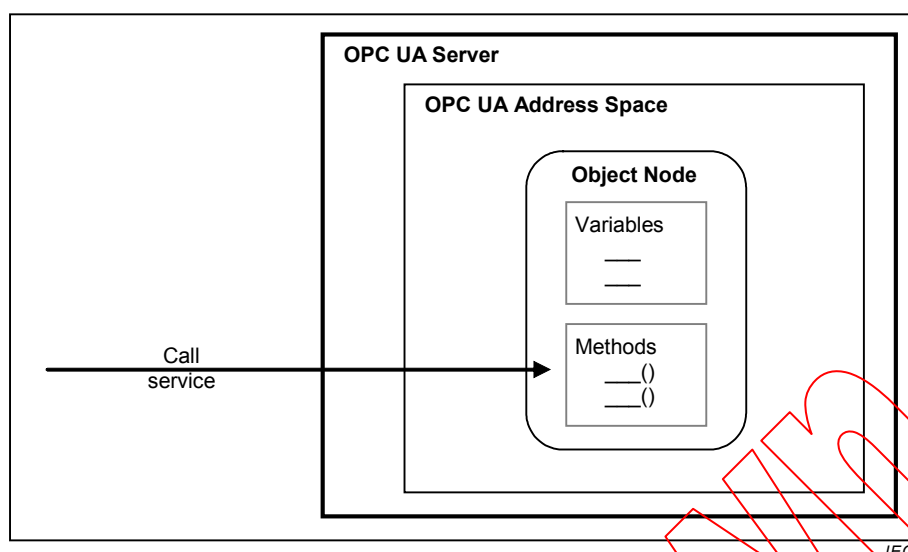


Figure 7 – Method Service Set

The *MonitoredItem Service Set* and the *Subscription Service Set*, illustrated in Figure 8, are used together to subscribe to *Nodes* in the OPC UA *AddressSpace*.

The *MonitoredItem Service Set* defines *Services* that allow *Clients* to create, modify, and delete *MonitoredItems* used to monitor *Attributes* for value changes and *Objects* for *Events*.

These *Notifications* are queued for transfer to the *Client* by *Subscriptions*.

The *Subscription Service Set* defines *Services* that allow *Clients* to create, modify and delete *Subscriptions*. *Subscriptions* send *Notifications* generated by *MonitoredItems* to the *Client*. *Subscription Services* also provide for *Client* recovery from missed *Messages* and communication failures.

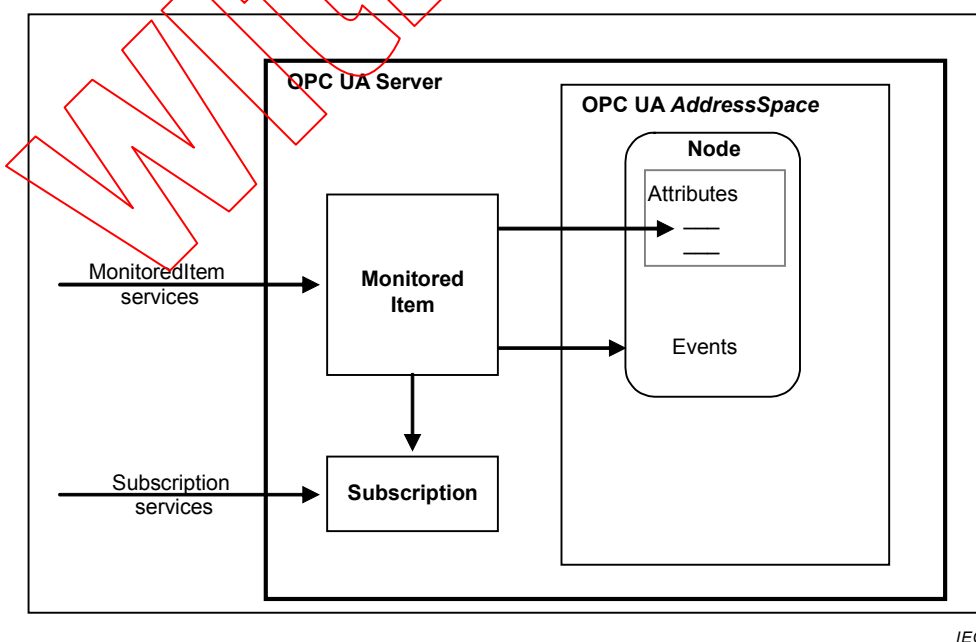


Figure 8 – MonitoredItem and Subscription Service Sets

4.2 Request/response Service procedures

Request/response *Service* procedures describe the processing of requests received by the *Server*, and the subsequent return of responses. The procedures begin with the requesting *Client* submitting a *Service* request *Message* to the *Server*.

Upon receipt of the request, the *Server* processes the *Message* in two steps. In the first step, it attempts to decode and locate the *Service* to execute. The error handling for this step is specific to the communication technology used and is described in IEC 62541-6.

If it succeeds, then it attempts to access each operation identified in the request and perform the requested operation. For each operation in the request, it generates a separate success/failure code that it includes in a positive response *Message* along with any data that is to be returned.

To perform these operations, both the *Client* and the *Server* may make use of the API of a *Communication Stack* to construct and interpret *Messages* and to access the requested operation.

The implementation of each service request or response handling shall check that each service parameter lies within the specified range for that parameter.

5 Service Sets

5.1 General

This clause defines the OPC UA *Service Sets* and their *Services*. Clause 7 contains the definitions of common parameters used by these *Services*. Clause 6.2 describes auditing requirements for all services.

Whether or not a *Server* supports a *Service Set*, or a *Service* within a *Service Set*, is defined by its *Profile*. *Profiles* are described in IEC 62541-7.

5.2 Service request and response header

Each *Service* request has a *RequestHeader* and each *Service* response has a *ResponseHeader*.

The *RequestHeader* structure is defined in 7.26 and contains common request parameters such as *authenticationToken*, *timestamp* and *requestHandle*.

The *ResponseHeader* structure is defined in 7.27 and contains common response parameters such as *serviceResult* and *diagnosticInfo*.

5.3 Service results

Service results are returned at two levels in OPC UA responses, one that indicates the status of the *Service* call, and the other that indicates the status of each operation requested by the *Service*.

Service results are defined via the *StatusCode* (see 7.33).

The status of the *Service* call is represented by the *serviceResult* contained in the *ResponseHeader* (see 7.27). The mechanism for returning this parameter is specific to the communication technology used to convey the *Service* response and is defined in IEC 62541-6.

The status of individual operations in a request is represented by individual *StatusCodes*.

The following cases define the use of these parameters.

- a) A bad code is returned in *serviceResult* if the *Service* itself failed. In this case, a *ServiceFault* is returned. The *ServiceFault* is defined in 7.28.
- b) The good code is returned in *serviceResult* if the *Service* fully or partially succeeded. In this case, other response parameters are returned. The *Client* shall always check the response parameters, especially all *StatusCodes* associated with each operation. These *StatusCodes* may indicate bad or uncertain results for one or more operations requested in the *Service* call.

All *Services* with arrays of operations in the request shall return a bad code in the *serviceResult* if the array is empty.

The *Services* define various specific *StatusCodes* and a *Server* shall use these specific *StatusCodes* as described in the *Service*. A *Client* should be able to handle these *Service* specific *StatusCodes*. In addition, a *Client* shall expect other common *StatusCodes* defined in Table 165 and Table 166. Additional details for *Client* handling of specific *StatusCodes* may be defined in IEC 62541-7.

If the *Server* discovers, through some out-of-band mechanism that the application or user credentials used to create a *Session* or *SecureChannel* have been compromised, then the *Server* should immediately terminate all sessions and channels that use those credentials. In this case, the *Service* result code should be either *Bad_IdentityTokenRejected* or *Bad_CertificateUntrusted*.

Message parsing can fail due to syntax errors or if data contained within the message exceeds ranges supported by the receiver. When this happens messages shall be rejected by the receiver. If the receiver is a *Server* then it shall return a *ServiceFault* with result code of *Bad_DecodingError* or *Bad_EncodingLimitsExceeded*. If the receiver is the *Client* then the *Communication Stack* should report these errors to the *Client* application.

Many applications will place limits on the size of messages and/or data elements contained within these messages. For example, a *Server* may reject requests containing string values longer than a certain length. These limits are typically set by administrators and apply to all connections between a *Client* and a *Server*.

Clients that receive *Bad_EncodingLimitsExceeded* faults from the *Server* will likely have to reformulate their requests. The administrator may need to increase the limits for the *Client* if it receives a response from the *Server* with this fault.

In some cases, parsing errors are fatal and it is not possible to return a fault. For example, the incoming message could exceed the buffer capacity of the receiver. In these cases, these errors may be treated as a communication fault which requires the *SecureChannel* to be re-established (see 5.5).

The *Client* and *Server* reduce the chances of a fatal error by exchanging their message size limits in the *CreateSession* service. This will allow either party to avoid sending a message that causes a communication fault. The *Server* should return a *Bad_ResponseTooLarge* fault if a serialized response message exceeds the message size specified by the *Client*. Similarly, the *Client Communication Stack* should report a *Bad_RequestTooLarge* error to the application before sending a message that exceeds the *Server's* limit.

Note that the message size limits only apply to the raw message body and do not include headers or the effect of applying any security. This means that a message body that is smaller than the specified maximum could still cause a fatal error.

5.4 Discovery Service Set

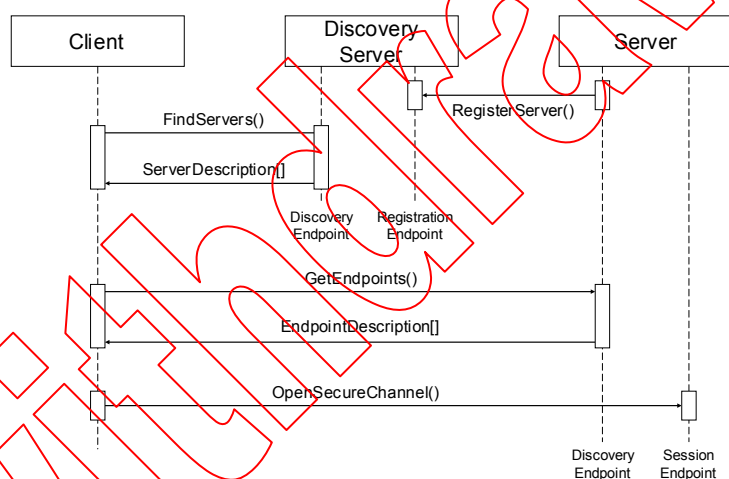
5.4.1 Overview

This *Service Set* defines *Services* used to discover the *Endpoints* implemented by a *Server* and to read the security configuration for those *Endpoints*. The *Discovery Services* are implemented by individual *Servers* and by dedicated *Discovery Servers*. IEC 62541-12¹ describes how to use the *Discovery Services* with dedicated *Discovery Servers*.

Every *Server* shall have a *Discovery Endpoint* that *Clients* can access without establishing a *Session*. This *Endpoint* may or may not be the same *Session Endpoint* that *Clients* use to establish a *SecureChannel*. *Clients* read the security information necessary to establish a *SecureChannel* by calling the *GetEndpoints* Service on the *Discovery Endpoint*.

In addition, *Servers* may register themselves with a well known *Discovery Server* using the *RegisterServer* Service. *Clients* can later discover any registered *Servers* by calling the *FindServers* Service on the *Discovery Server*.

The complete discovery process is illustrated in Figure 9.



IEC

Figure 9 – Discovery process

The URL for a *Discovery Endpoint* shall provide all of the information that the *Client* needs to connect to the *Discovery Endpoint*.

Once a *Client* retrieves the *Endpoints*, the *Client* can save this information and use it to connect directly to the *Server* again without going through the discovery process. If the *Client* finds that it cannot connect then the *Server* configuration may have changed and the *Client* needs to go through the discovery process again.

Discovery Endpoints shall not require any message security, but it may require transport layer security. In production systems, Administrators may disable discovery for security reasons and *Clients* shall rely on cached *EndpointDescriptions*. To provide support for systems with disabled *Discovery Services* *Clients* shall allow Administrators to manually update the *EndpointDescriptions* used to connect to a *Server*. *Servers* shall allow Administrators to disable the *Discovery Endpoint*.

¹ Under consideration.

A *Client* shall be careful when using the information returned from a *Discovery Endpoint* since it has no security. A *Client* does this by comparing the information returned from the *Discovery Endpoint* to the information returned in the *CreateSession* response. A *Client* shall verify that:

- a) The *ApplicationUri* specified in the *Server Certificate* is the same as the *ApplicationUri* provided in the *EndpointDescription*.
- b) The *Server Certificate* returned in *CreateSession* response is the same as the *Certificate* used to create the *SecureChannel*.
- c) The *EndpointDescriptions* returned from the *Discovery Endpoint* are the same as the *EndpointDescriptions* returned in the *CreateSession* response.

If the *Client* detects that one of the above requirements is not fulfilled, then the *Client* shall close the *SecureChannel* and report an error.

A *Client* shall verify the *HostName* specified in the *Server Certificate* is the same as the *HostName* contained in the *endpointUrl* provided in the *EndpointDescription* returned by *CreateSession*. If there is a difference then the *Client* shall report the difference and may close the *SecureChannel*. Servers shall add all possible *HostNames* like *MyHost* and *MyHost.local* into the *Server Certificate*. This includes IP addresses of the host or the *HostName* exposed by a NAT router used to connect to the *Server*.

5.4.2 FindServers

5.4.2.1 Description

This *Service* returns the *Servers* known to a *Server* or *Discovery Server*. The behaviour of *Discovery Servers* is described in detail in IEC 62541-12².

The *Client* may reduce the number of results returned by specifying filter criteria. A *Discovery Server* returns an empty list if no *Servers* match the criteria specified by the client. The filter criteria supported by this *Service* are described in 5.4.2.2.

Every *Server* shall provide a *Discovery Endpoint* that supports this *Service*. The *Server* shall always return a record that describes itself, however in some cases more than one record may be returned. *Gateway Servers* shall return a record for each *Server* that they provide access to plus (optionally) a record that allows the *Gateway Server* to be accessed as an ordinary OPC UA *Server*. Non-transparent redundant *Servers* shall provide a record for each *Server* in the redundant set.

Every *Server* shall have a globally unique identifier called the *ServerUri*. This identifier should be a fully qualified domain name; however, it may be a GUID or similar construct that ensures global uniqueness. The *ServerUri* returned by this *Service* shall be the same value that appears in index 0 of the *ServerArray* property (see IEC 62541-5). The *ServerUri* is returned as the *applicationUri* field in the *ApplicationDescription* (see 7.1)

Every *Server* shall also have a human readable identifier called the *ServerName* which is not necessarily globally unique. This identifier may be available in multiple locales.

A *Server* may have multiple *HostNames*. For this reason, the *Client* shall pass the URL it used to connect to the *Endpoint* to this *Service*. The implementation of this *Service* shall use this information to return responses that are accessible to the *Client* via the provided URL.

This *Service* shall not require any message security but it may require transport layer security.

² Under consideration.

Some *Servers* may be accessed via a *Gateway Server* and shall have a value specified for *gatewayServerUri* in their *ApplicationDescription* (see 7.1). The *discoveryUrls* provided in *ApplicationDescription* shall belong to the *Gateway Server*. Some *Discovery Servers* may return multiple records for the same *Server* if that *Server* can be accessed via multiple paths.

This *Service* can be used without security and it is therefore vulnerable to Denial Of Service (DOS) attacks. A *Server* should minimize the amount of processing required to send the response for this *Service*. This can be achieved by preparing the result in advance. The *Server* should also add a short delay before starting processing of a request during high traffic conditions.

The *DiscoveryUrl* returned by this *Service* is ambiguous if there are multiple *TransportProfiles* (e.g. UA XML or UA Binary encoding) associated with the URL scheme. *Clients* that support multiple *TransportProfiles* should attempt to use alternate *TransportProfiles* if the first choice does not succeed.

5.4.2.2 Parameters

Table 3 defines the parameters for the *Service*.

Table 3 – FindServers Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>authenticationToken</i> is always omitted. The <i>authenticationToken</i> shall be ignored if it is provided. The type <i>RequestHeader</i> is defined in 7.26.
endpointUrl	String	The network address that the <i>Client</i> used to access the <i>Discovery Endpoint</i> . The <i>Server</i> uses this information for diagnostics and to determine what URLs to return in the response. The <i>Server</i> should return a suitable default URL if it does not recognize the <i>HostName</i> in the URL.
localeIds []	LocaleId	List of locales to use. The server should return the <i>ServerName</i> using one of locales specified. If the server supports more than one of the requested locales then the server shall use the locale that appears first in this list. If the server does not support any of the requested locales it chooses an appropriate default locale. The server chooses an appropriate default locale if this list is empty.
serverUri []	String	List of servers to return. All known servers are returned if the list is empty. A <i>serverUri</i> matches the <i>applicationUri</i> from the <i>ApplicationDescription</i> defined in 7.1.
Response		
responseHeader	ResponseHeader	Common response parameters. The <i>ResponseHeader</i> type is defined in 7.27.
servers []	ApplicationDescription	List of <i>Servers</i> that meet criteria specified in the request. This list is empty if no servers meet the criteria. The <i>ApplicationDescription</i> type is defined in 7.1.

5.4.2.3 Service results

Common *StatusCodes* are defined in Table 165.

5.4.3 GetEndpoints

5.4.3.1 Description

This *Service* returns the *Endpoints* supported by a *Server* and all of the configuration information required to establish a *SecureChannel* and a *Session*.

This *Service* shall not require any message security but it may require transport layer security.

A *Client* may reduce the number of results returned by specifying filter criteria based on *LocaleIds* and *Transport Profile* URIs. The *Server* returns an empty list if no *Endpoints* match the criteria specified by the client. The filter criteria supported by this *Service* are described in 5.4.3.2.

A *Server* may support multiple security configurations for the same *Endpoint*. In this situation, the *Server* shall return separate *EndpointDescription* records for each available configuration. *Clients* should treat each of these configurations as distinct *Endpoints* even if the physical URL happens to be the same.

The security configuration for an *Endpoint* has four components:

- Server Application Instance Certificate
- Message Security Mode
- Security Policy
- Supported User Identity Tokens

The *ApplicationInstanceCertificate* is used to secure the *OpenSecureChannel* request (see 5.5.2). The *MessageSecurityMode* and the *SecurityPolicy* tell the *Client* how to secure messages sent via the *SecureChannel*. The *UserIdentityTokens* tell the client which type of user credentials shall be passed to the *Server* in the *ActivateSession* request (see 5.6.3).

If the *securityPolicyUri* is NONE and none of the *UserTokenPolicies* requires encryption, the *Client* shall ignore the *ApplicationInstanceCertificate*.

Each *EndpointDescription* also specifies a URI for the *Transport Profile* that the *Endpoint* supports. The *Transport Profiles* specify information such as message encoding format and protocol version and are defined in IEC 62541-7. *Clients* shall fetch the *Server's SoftwareCertificates* if they want to discover the complete list of *Profiles* supported by the *Server* (see 7.30).

Messages are secured by applying standard cryptography algorithms to the messages before they are sent over the network. The exact set of algorithms used depends on the *SecurityPolicy* for the *Endpoint*. IEC 62541-7 defines *Profiles* for common *SecurityPolicies* and assigns a unique URI to them. It is expected that applications have built in knowledge of the *SecurityPolicies* that they support, as a result, only the Profile URI for the *SecurityPolicy* is specified in the *EndpointDescription*. A *Client* cannot connect to an *Endpoint* that does not support a *SecurityPolicy* that it recognizes.

An *EndpointDescription* may specify that the message security mode is NONE. This configuration is not recommended unless the applications are communicating on a physically isolated network where the risk of intrusion is extremely small. If the message security is NONE then it is possible for *Clients* to deliberately or accidentally hijack *Sessions* created by other *Clients*.

A *Server* may have multiple *HostNames*. For this reason, the *Client* shall pass the URL it used to connect to the *Endpoint* to this *Service*. The implementation of this *Service* shall use this information to return responses that are accessible to the *Client* via the provided URL.

This *Service* can be used without security and it is therefore vulnerable to Denial Of Service (DOS) attacks. A *Server* should minimize the amount of processing required to send the response for this *Service*. This can be achieved by preparing the result in advance. The *Server* should also add a short delay before starting processing of a request during high traffic conditions.

Some of the *EndpointDescriptions* returned in a response shall specify the *Endpoint* information for a *Gateway Server* that can be used to access another *Server*. In these situations, the *gatewayServerUri* is specified in the *EndpointDescription* and all security

checks used to verify *Certificates* shall use the *gatewayServerUri* (see 6.1.3) instead of the *serverUri*.

To connect to a *Server* via the gateway the *Client* shall first establish a *SecureChannel* with the *Gateway Server*. Then the *Client* shall call the *CreateSession* service and pass the *serverUri* specified in the *EndpointDescription* to the *Gateway Server*. The *Gateway Server* shall then connect to the underlying *Server* on behalf of the *Client*. The process of connecting to a *Server* via a *Gateway Server* is illustrated in Figure 10.

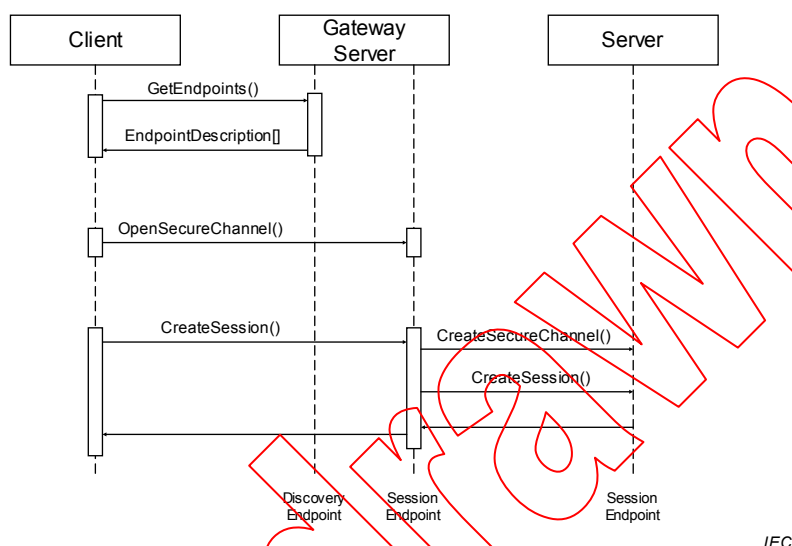


Figure 10 – Using a Gateway Server

5.4.3.2 Parameters

Table 4 defines the parameters for the *Service*.

Table 4 – GetEndpoints Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>authenticationToken</i> is always omitted. The <i>authenticationToken</i> shall be ignored if it is provided. The type <i>RequestHeader</i> is defined in 7.26.
endpointUrl	String	The network address that the <i>Client</i> used to access the <i>Discovery Endpoint</i> . The <i>Server</i> uses this information for diagnostics and to determine what URLs to return in the response. The <i>Server</i> should return a suitable default URL if it does not recognize the <i>HostName</i> in the URL.
localeIds []	LocaleId	List of locales to use. Specifies the locale to use when returning human readable strings. This parameter is described in 5.4.2.2.
profileUris []	String	List of <i>Transport Profile</i> that the returned <i>Endpoints</i> shall support. IEC 62541-7 defines URIs for the <i>Transport Profiles</i> . All <i>Endpoints</i> are returned if the list is empty.
Response		
responseHeader	ResponseHeader	Common response parameters. The <i>ResponseHeader</i> type is defined in 7.27.
Endpoints []	EndpointDescription	List of <i>Endpoints</i> that meet criteria specified in the request. This list is empty if no <i>Endpoints</i> meet the criteria. The <i>EndpointDescription</i> type is defined in 7.9.

5.4.3.3 Service Results

Common *StatusCodes* are defined in Table 165.

5.4.4 RegisterServer

5.4.4.1 Description

This *Service* registers a *Server* with a *Discovery Server*. This *Service* will be called by a *Server* or a separate configuration utility. *Clients* will not use this *Service*.

A *Server* shall establish a *SecureChannel* with the *Discovery Server* before calling this *Service*. The *SecureChannel* is described in 5.5. The *Administrator* of the *Server* shall provide the *Server* with an *EndpointDescription* for the *Discovery Server* as part of the configuration process. *Discovery Servers* shall reject registrations if the *serverUri* provided does not match the *applicationUri* in *Server Certificate* used to create the *SecureChannel*.

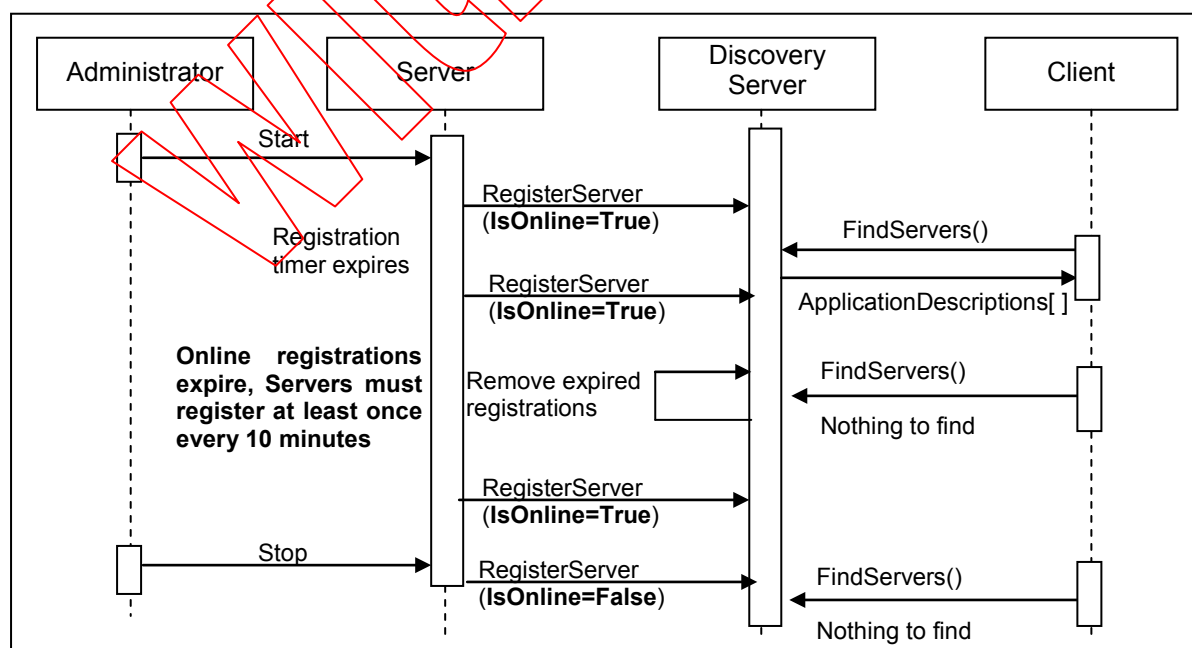
A *Server* only provides its *serverUri* and the URLs of the *Discovery Endpoints* to the *Discovery Server*. *Clients* shall use the *GetEndpoints* service to fetch the most up to date configuration information directly from the *Server*.

The *Server* shall provide a localized name for itself in all locales that it supports.

Servers shall be able to register themselves with a *Discovery Server* running on the same machine. The exact mechanisms depend on the *Discovery Server* implementation and are described in IEC 62541-6.

There are two types of *Server* applications: those which are manually launched including a start by the operating system at boot and those that are automatically launched when a *Client* attempts to connect. The registration process that a *Server* shall use depends on which category it falls into.

The registration process for manually launched *Servers* is illustrated in Figure 11.



IEC

Figure 11 – The Registration Process – Manually Launched Servers

The registration process for automatically launched *Servers* is illustrated in Figure 12.

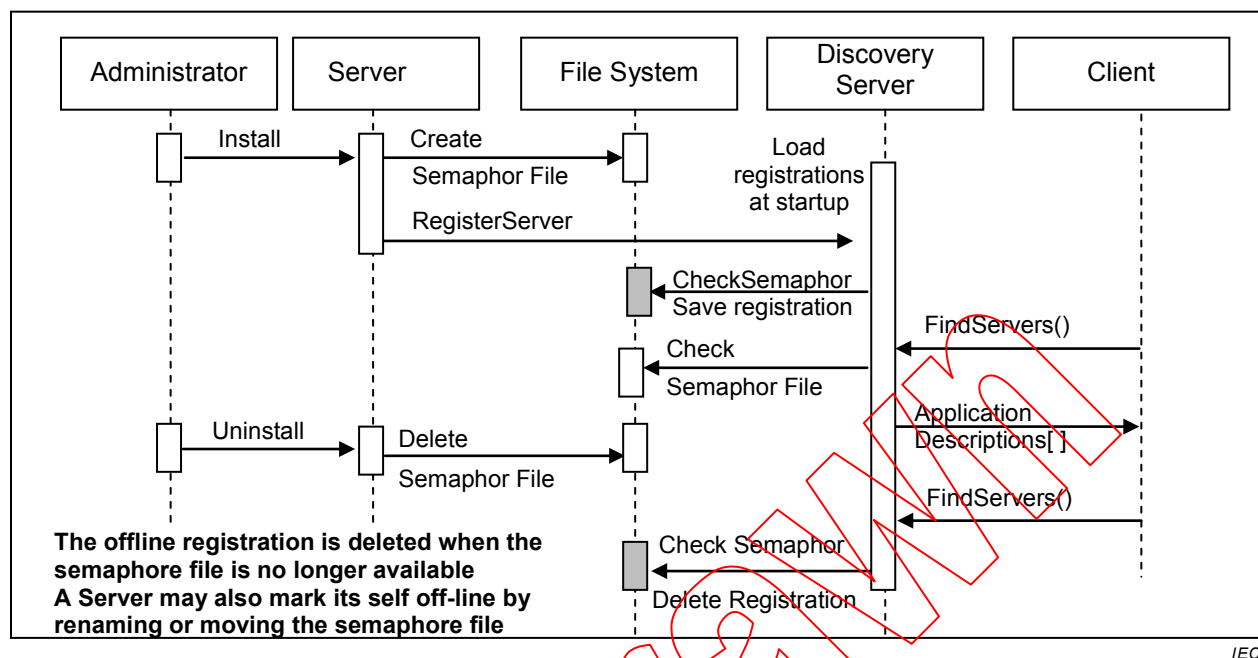


Figure 12 – The Registration Process – Automatically Launched Servers

The registration process is designed to be platform independent, robust and able to minimize problems created by configuration errors. For that reason, *Servers* shall register themselves more than once.

Under normal conditions, manually launched *Servers* shall periodically register with the *Discovery Server* as long as they are able to receive connections from *Clients*. If a *Server* goes offline then it shall register itself once more and indicate that it is going offline. The registration frequency should be configurable; however, the maximum is 10 minutes. If an error occurs during registration (e.g. the *Discovery Server* is not running) then the *Server* shall periodically re-attempt registration. The frequency of these attempts should start at 1 second but gradually increase until the registration frequency is the same as what it would be if no errors occurred. The recommended approach would be to double the period of each attempt until reaching the maximum.

When an automatically launched *Server* (or its install program) registers with the *Discovery Server* it shall provide a path to a semaphore file which the *Discovery Server* can use to determine if the *Server* has been uninstalled from the machine. The *Discovery Server* shall have read access to the file system that contains the file.

5.4.4.2 Parameters

Table 5 defines the parameters for the *Service*.

Table 5 – RegisterServer Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>authenticationToken</i> is always omitted. The type <i>RequestHeader</i> is defined in 7.26.
Server	RegisteredServer	The server to register. This structure is defined in-line with the following indented items.
serverUri	String	The globally unique identifier for the <i>Server</i> instance. The <i>serverUri</i> matches the <i>applicationUri</i> from the <i>ApplicationDescription</i> defined in 7.1.
productUri	String	The globally unique identifier for the <i>Server</i> product.
serverNames []	LocalizedText	A list of localized descriptive names for the <i>Server</i> . The list shall have at least one valid entry.
serverType	Enum ApplicationType	The type of application. The enumeration values are defined in Table 163. The value "CLIENT_1" (The application is a <i>Client</i>) is not allowed. The <i>Service</i> result shall be <i>Bad_InvalidArgument</i> in this case.
gatewayServerUri	String	The URI of the <i>Gateway Server</i> associated with the <i>discoveryUris</i> . This value is only specified by <i>Gateway Servers</i> that wish to register the <i>Servers</i> that they provide access to. For <i>Servers</i> that do not act as a <i>Gateway Server</i> this parameter shall be null.
discoveryUris []	String	A list of <i>Discovery Endpoints</i> for the <i>Server</i> . The list shall have at least one valid entry.
semaphoreFilePath	String	The path to the semaphore file used to identify an automatically-launched server instance; Manually-launched servers will not use this parameter. If a Semaphore file is provided, the <i>isOnline</i> flag is ignored. If a Semaphore file is provided and exists, the <i>LocalDiscoveryServer</i> shall save the registration information in a persistent data store that it reads whenever the <i>LocalDiscoveryServer</i> starts. If a Semaphore file is specified but does not exist the <i>Discovery Server</i> shall remove the registration from any persistent data store. If the <i>Server</i> has registered with a <i>semaphoreFilePath</i> , the <i>DiscoveryServer</i> shall check that this file exists before returning the <i>ApplicationDescription</i> to the client. If the <i>Server</i> did not register with a <i>semaphoreFilePath</i> (it is null or empty) then the <i>DiscoveryServer</i> does not attempt to verify the existence of the file before returning the <i>ApplicationDescription</i> to the client.
isOnline	Boolean	True if the <i>Server</i> is currently able to accept connections from <i>Clients</i> . The <i>DiscoveryServer</i> shall return <i>ApplicationDescriptions</i> to the <i>Client</i> . The <i>Server</i> is expected to periodically re-register with the <i>DiscoveryServer</i> . False if the <i>Server</i> is currently unable to accept connections from <i>Clients</i> . The <i>DiscoveryServer</i> shall NOT return <i>ApplicationDescriptions</i> to the <i>Client</i> . This parameter is ignored if a <i>semaphoreFilePath</i> is provided.
Response		
ResponseHeader	ResponseHeader	Common response parameters. The type <i>ResponseHeader</i> is defined in 7.27.

5.4.4.3 Service Results

Table 6 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 6 – RegisterServer Service Result Codes

Symbolic Id	Description
Bad_InvalidArgument	See Table 165 for the description of this result code.
Bad_ServerUriInvalid	See Table 165 for the description of this result code.
Bad_ServerNameMissing	No <i>ServerName</i> was specified.
Bad_DiscoveryUriMissing	No discovery URL was specified.
Bad_SemaphoreFileMissing	The semaphore file specified is not valid.

5.5 SecureChannel Service Set

5.5.1 Overview

This *Service Set* defines *Services* used to open a communication channel that ensures the confidentiality and *Integrity* of all *Messages* exchanged with the *Server*. The base concepts for OPC UA security are defined in IEC TR 62541-2.

The *SecureChannel Services* are unlike other *Services* because they are not implemented directly by the *OPC UA Application*. Instead, they are provided by the *Communication Stack* on which the *OPC UA Application* is built. For example, an OPC UA *Server* may be built on a SOAP stack that allows applications to establish a *SecureChannel* using the WS Secure Conversation specification. In these cases, the *OPC UA Application* shall verify that the *Message* it received was in the context of a WS Secure Conversation. IEC 62541-6 describes how the *SecureChannel Services* are implemented.

A *SecureChannel* is a long-running logical connection between a single *Client* and a single *Server*. This channel maintains a set of keys known only to the *Client* and *Server*, which are used to authenticate and encrypt *Messages* sent across the network. The *SecureChannel Services* allow the *Client* and *Server* to securely negotiate the keys to use.

An *EndpointDescription* tells a *Client* how to establish a *SecureChannel* with a given *Endpoint*. A *Client* may obtain the *EndpointDescription* from a *Discovery Server*, via some non-UA defined directory server or from its own configuration.

The exact algorithms used to authenticate and encrypt *Messages* are described in the *SecurityPolicy* field of the *EndpointDescription*. A *Client* shall use these algorithms when it creates a *SecureChannel*.

It should be noted that some *SecurityPolicies* defined in IEC 62541-7 will turn off authentication and encryption resulting in a *SecureChannel* that provides no security.

When a *Client* and *Server* are communicating via a *SecureChannel*, they shall verify that all incoming *Messages* have been signed and encrypted according to the requirements specified in the *EndpointDescription*. An *OPC UA Application* shall not process any *Message* that does not conform to these requirements.

The relationship between the *SecureChannel* and the *OPC UA Application* depends on the implementation technology. IEC 62541-6 defines any requirements that depend on the technology used.

The correlation between the *OPC UA Application Session* and the *SecureChannel* is illustrated in Figure 13. The *Communication Stack* is used by the *OPC UA Applications* to exchange *Messages*. In the first step, the *SecureChannel Services* are used to establish a *SecureChannel* between the two *Communication Stacks* which allows the secure exchange of *Messages*. In the second step, the *OPC UA Applications* use the *Session Service Set* to establish an *OPC UA Application Session*.

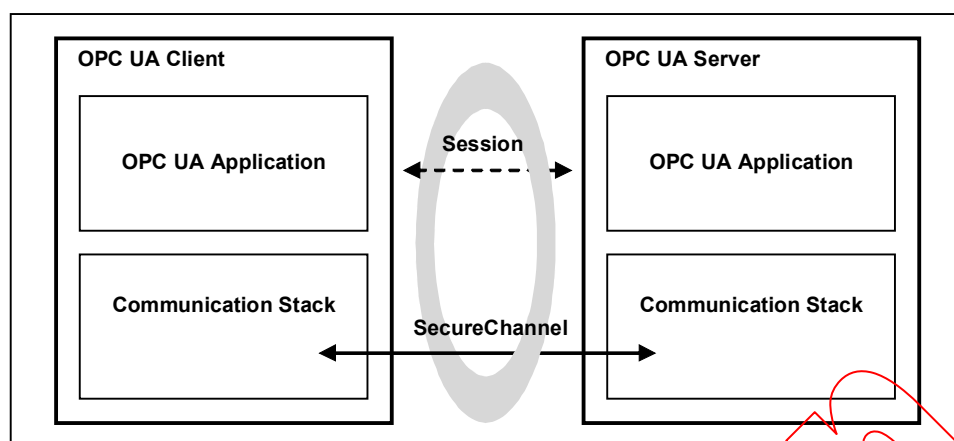


Figure 13 – SecureChannel and Session Services

Once a *Client* has established a *Session* it may wish to access the *Session* from a different *SecureChannel*. The *Client* can do this by validating the new *SecureChannel* with the *ActivateSession Service* described in 5.6.3.

If a *Server* acts as a *Client* to other *Servers*, which is commonly referred to as *Server chaining*, then the *Server* shall be able to maintain user level security. By this we mean that the user identity should be passed to the underlying server or it should be mapped to an appropriate user identity in the underlying server. It is unacceptable to ignore user level security. This is required to ensure that security is maintained and that a user does not obtain information that they should not have access to. Whenever possible a *Server* should impersonate the original *Client* by passing the original *Client's* user identity to the underlying *Server* when it calls the *ActivateSession Service*. If impersonation is not an option then the *Server* shall map the original *Client's* user identity onto a new user identity which the underlying *Server* does recognize.

5.5.2 OpenSecureChannel

5.5.2.1 Description

This *Service* is used to open or renew a *SecureChannel* that can be used to ensure *Confidentiality* and *Integrity* for *Message* exchange during a *Session*. This *Service* requires the *Communication Stack* to apply the various security algorithms to the *Messages* as they are sent and received. Specific implementations of this *Service* for different *Communication Stacks* are described in IEC 62541-6.

Each *SecureChannel* has a globally-unique identifier and is valid for a specific combination of *Client* and *Server* application instances. Each channel contains one or more *SecurityTokens* that identify a set of cryptography keys that are used to encrypt and authenticate *Messages*. *SecurityTokens* also have globally-unique identifiers which are attached to each *Message* secured with the token. This allows an authorized receiver to know how to decrypt and verify the *Message*.

SecurityTokens have a finite lifetime negotiated with this *Service*. However, differences between the system clocks on different machines and network latencies mean that valid *Messages* could arrive after the token has expired. To prevent valid *Messages* from being discarded, the applications should do the following:

- a) *Clients* should request a new *SecurityToken* after 75 % of its lifetime has elapsed. This should ensure that *Clients* will receive the new *SecurityToken* before the old one actually expires.

- b) *Servers* shall use the existing *SecurityToken* to secure outgoing *Messages* until the *SecurityToken* expires or the *Server* receives a *Message* secured with a new *SecurityToken*. This should ensure that *Clients* do not reject *Messages* secured with the new *SecurityToken* that arrive before the *Client* receives the new *SecurityToken*.
- c) *Clients* should accept *Messages* secured by an expired *SecurityToken* for up to 25 % of the token lifetime. This should ensure that *Messages* sent by the *Server* before the token expired are not rejected because of network delays.

Each *SecureChannel* exists until it is explicitly closed or until the last token has expired and the overlap period has elapsed. A *Server* application should limit the number of *SecureChannels*. To protect against misbehaving *Clients* and denial of service attacks, the *Server* shall close the oldest *SecureChannel* that has no *Session* assigned before reaching the maximum number of supported *SecureChannels*.

The *OpenSecureChannel* request and response *Messages* shall be signed with the sender's *Certificate*. These *Messages* shall always be encrypted. If the transport layer does not provide encryption, then these *Messages* shall be encrypted with the receiver's *Certificate*. These requirements for *OpenSecureChannel* only apply if the *securityPolicyUri* is not *None*.

The *Certificates* used in the *OpenSecureChannel* service shall be the *Application Instance Certificates*. *Clients* and *Servers* shall verify that the same *Certificates* were used in the *CreateSession* and *ActivateSession* services. *Certificates* are not provided and shall not be verified if the *securityPolicyUri* is *None*.

If the *securityPolicyUri* is not *None*, a *Client* shall verify the *HostName* specified in the *Server Certificate* is the same as the *HostName* contained in the *endpointUrl*. If there is a difference then the *Client* shall report the difference and may choose to not open the *SecureChannel*. *Servers* shall add all possible *HostNames* like *MyHost* and *MyHost.local* into the *Server Certificate*. This includes IP addresses of the host or the *HostName* exposed by a NAT router used to connect to the *Server*.

Clients should be prepared to replace the *HostName* returned in the *EndPointDescription* with the *HostName* or the IP addresses they used to call *GetEndpoints*.

5.5.2.2 Parameters

Table 7 defines the parameters for the *Service*.

Unlike other *Services*, the parameters for this *Service* provide only an abstract definition. The concrete representation on the network depends on the mappings defined in IEC 62541-6.

Table 7 – OpenSecureChannel Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>authenticationToken</i> is always omitted. The type <i>RequestHeader</i> is defined in 7.26.
clientCertificate	ApplicationInstanceCertificate	A <i>Certificate</i> that identifies the <i>Client</i> . The <i>OpenSecureChannel</i> request shall be signed with this <i>Certificate</i> . The <i>ApplicationInstanceCertificate</i> type is defined in 7.2. If the <i>securityPolicyUri</i> is None, the <i>Server</i> shall ignore the <i>ApplicationInstanceCertificate</i> .
requestType	Enum SecurityToken RequestType	The type of <i>SecurityToken</i> request: An enumeration that shall be one of the following: ISSUE_0 creates a new <i>SecurityToken</i> for a new <i>SecureChannel</i> . RENEW_1 creates a new <i>SecurityToken</i> for an existing <i>SecureChannel</i> .
secureChannelId	ByteString	The identifier for the <i>SecureChannel</i> that the new token should belong to. This parameter shall be null when creating a new <i>SecureChannel</i> .
securityMode	Enum MessageSecurityMode	The type of security to apply to the messages. The type <i>MessageSecurityMode</i> type is defined in 7.14. A <i>SecureChannel</i> may have to be created even if the <i>securityMode</i> is NONE. The exact behaviour depends on the mapping used and is described in the IEC 62541-6.
securityPolicyUri	String	The URI for <i>SecurityPolicy</i> to use when securing messages sent over the <i>SecureChannel</i> . The set of known URIs and the <i>SecurityPolicies</i> associated with them are defined in IEC 62541-7.
clientNonce	ByteString	A random number that shall not be used in any other request. A new <i>clientNonce</i> shall be generated for each time a <i>SecureChannel</i> is renewed. This parameter shall have a length equal to key size used for the symmetric encryption algorithm that is identified by the <i>securityPolicyUri</i> .
requestedLifetime	Duration	The requested lifetime, in milliseconds, for the new <i>SecurityToken</i> . It specifies when the <i>Client</i> expects to renew the <i>SecureChannel</i> by calling the <i>OpenSecureChannel Service</i> again. If a <i>SecureChannel</i> is not renewed, then all <i>Messages</i> sent using the current <i>SecurityTokens</i> shall be rejected by the receiver.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> type definition).
securityToken	ChannelSecurityToken	Describes the new <i>SecurityToken</i> issued by the <i>Server</i> . This structure is defined in-line with the following indented items.
channelId	ByteString	A unique identifier for the <i>SecureChannel</i> . This is the identifier that shall be supplied whenever the <i>SecureChannel</i> is renewed.
tokenId	ByteString	A unique identifier for a single <i>SecurityToken</i> within the channel. This is the identifier that shall be passed with each <i>Message</i> secured with the <i>SecurityToken</i> .
createdAt	UtcTime	The time when the <i>SecurityToken</i> was created.
revisedLifetime	Duration	The lifetime of the <i>SecurityToken</i> in milliseconds. The UTC expiration time for the token may be calculated by adding the lifetime to the <i>createdAt</i> time.
serverNonce	ByteString	A random number that shall not be used in any other request. A new <i>serverNonce</i> shall be generated for each time a <i>SecureChannel</i> is renewed. This parameter shall have a length equal to key size used for the symmetric encryption algorithm that is identified by the <i>securityPolicyUri</i> .

5.5.2.3 Service results

Table 8 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 8 – OpenSecureChannel Service Result Codes

Symbolic Id	Description
Bad_SecurityChecksFailed	See Table 165 for the description of this result code.
Bad_CertificateTimeInvalid	See Table 165 for the description of this result code.
Bad_CertificateIssuerTimeInvalid	See Table 165 for the description of this result code.
Bad_CertificateHostNameInvalid	See Table 165 for the description of this result code.
Bad_CertificateUriInvalid	See Table 165 for the description of this result code.
Bad_CertificateUseNotAllowed	See Table 165 for the description of this result code.
Bad_CertificateIssuerUseNotAllowed	See Table 165 for the description of this result code.
Bad_CertificateUntrusted	See Table 165 for the description of this result code.
Bad_CertificateRevocationUnknown	See Table 165 for the description of this result code.
Bad_CertificateIssuerRevocationUnknown	See Table 165 for the description of this result code.
Bad_CertificateRevoked	See Table 165 for the description of this result code.
Bad_CertificateIssuerRevoked	See Table 165 for the description of this result code.
Bad_RequestTypeInvalid	The security token request type is not valid.
Bad_SecurityModeRejected	The security mode does not meet the requirements set by the Server.
Bad_SecurityPolicyRejected	The security policy does not meet the requirements set by the Server.
Bad_SecureChannelIdInvalid	See Table 165 for the description of this result code.
Bad_NonceInvalid	See Table 165 for the description of this result code. A server shall check the minimum length of the client nonce and return this status if the length is below 32 bytes. A check for duplicated nonces can only be done in OpenSecureChannel calls with the request type RENEW_1.

5.5.3 CloseSecureChannel

5.5.3.1 Description

This *Service* is used to terminate a *SecureChannel*.

The request *Messages* shall be signed with the appropriate key associated with the current token for the *SecureChannel*.

5.5.3.2 Parameters

Table 9 defines the parameters for the *Service*.

Table 9 – CloseSecureChannel Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>sessionId</i> is always omitted. The type <i>RequestHeader</i> is defined in 7.26.
secureChannelId	ByteString	The identifier for the <i>SecureChannel</i> to close.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).

5.5.3.3 Service results

Table 10 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 10 – CloseSecureChannel Service Result Codes

Symbolic Id	Description
Bad_SecureChannelIdInvalid	See Table 165 for the description of this result code.

5.6 Session Service Set

5.6.1 Overview

This *Service Set* defines *Services* for an application layer connection establishment in the context of a *Session*.

5.6.2 CreateSession

5.6.2.1 Description

This *Service* is used by an OPC UA *Client* to create a *Session* and the *Server* returns two values which uniquely identify the *Session*. The first value is the *sessionId* which is used to identify the *Session* in the audit logs and in the *Server*'s address space. The second is the *authenticationToken* which is used to associate an incoming request with a *Session*.

Before calling this *Service*, the *Client* shall create a *SecureChannel* with the *OpenSecureChannel* *Service* to ensure the *Integrity* of all *Messages* exchanged during a *Session*. This *SecureChannel* has a unique identifier which the *Server* shall associate with the *authenticationToken*. The *Server* may accept requests with the *authenticationToken* only if they are associated with the same *SecureChannel* that was used to create the *Session*. The *Client* may associate a new *SecureChannel* with the *Session* by calling the *ActivateSession* method.

The *SecureChannel* is always managed by the *Communication Stack* which means it shall provide APIs which the *Server* can use to find out information about the *SecureChannel* used for any given request. The *Communication Stack* shall, at a minimum, provide the *SecurityPolicy* and *SecurityMode* used by the *SecureChannel*. It shall also provide a *SecureChannelId* which uniquely identifies the *SecureChannel* or the *Client Certificate* used to establish the *SecureChannel*. The *Server* uses one of these to identify the *SecureChannel* used to send a request. Clause 7.29 describes how to create the *authenticationToken* for different types of *Communication Stack*.

Depending upon on the *SecurityPolicy* and the *SecurityMode* of the *SecureChannel*, the exchange of *ApplicationInstanceCertificates* and *Nonces* may be optional and the signatures may be empty. See IEC 62541-7 for the definition of *SecurityPolicies* and the handling of these parameters.

The *Server* returns its *EndpointDescriptions* in the response. *Clients* use this information to determine whether the list of *EndpointDescriptions* returned from the *Discovery Endpoint* matches the *Endpoints* that the *Server* has. If there is a difference then the *Client* shall close the *Session* and report an error. The *Server* returns all *EndpointDescriptions* for the *serverUri* specified by the *Client* in the request. The *Client* only verifies *EndpointDescriptions* with a *transportProfileUri* that matches the *profileUri* specified in the original *GetEndpoints* request. A *Client* may skip this check if the *EndpointDescriptions* were provided by a trusted source such as the *Administrator*.

The *Session* created with this *Service* shall not be used until the *Client* calls the *ActivateSession* *Service* and provides its *SoftwareCertificates* and proves possession of its *Application Instance Certificate* and any user identity token that it provided.

The *SoftwareCertificates* parameter in the *Server* response is deprecated to reduce the message size for OPC UA Applications with limited resources. The *SoftwareCertificates* are provided in the *Server*'s *AddressSpace* as defined in IEC 62541-5. A *SoftwareCertificate* identifies the capabilities of the *Server* and also contains the list of OPC UA *Profiles* supported by the *Server*. OPC UA *Profiles* are defined in IEC 62541-7.

Additional *Certificates* issued by other organisations may be included to identify additional *Server* capabilities. Examples of these *Profiles* include support for specific information models and support for access to specific types of devices.

When a *Session* is created, the *Server* adds an entry for the *Client* in its *SessionDiagnosticsArray Variable*. See IEC 62541-5 for a description of this *Variable*.

Sessions are created to be independent of the underlying communications connection. Therefore, if a communications connection fails, the *Session* is not immediately affected. The exact mechanism to recover from an underlying communication connection error depends on the *SecureChannel* mapping as described in IEC 62541-6.

Sessions are terminated by the *Server* automatically if the *Client* fails to issue a *Service* request on the *Session* within the timeout period negotiated by the *Server* in the *CreateSession Service* response. This protects the *Server* against *Client* failures and against situations where a failed underlying connection cannot be re-established. *Clients* shall be prepared to submit requests in a timely manner to prevent the *Session* from closing automatically. *Clients* may explicitly terminate *Sessions* using the *CloseSession Service*.

When a *Session* is terminated, all outstanding requests on the *Session* are aborted and *Bad_SessionClosed StatusCodes* are returned to the *Client*. In addition, the *Server* deletes the entry for the *Client* from its *SessionDiagnosticsArray Variable* and notifies any other *Clients* who were subscribed to this entry.

If a *Client* invokes the *CloseSession Service* then all *Subscriptions* associated with the *Session* are also deleted if the *deleteSubscriptions* flag is set to TRUE. If a *Server* terminates a *Session* for any other reason, *Subscriptions* associated with the *Session*, are not deleted. Each *Subscription* has its own lifetime to protect against data loss in the case of a *Session* termination. In these cases, the *Subscription* can be reassigned to another *Client* before its lifetime expires.

Some *Servers*, such as aggregating *Servers*, also act as *Clients* to other *Servers*. These *Servers* typically support more than one system user, acting as their agent to the *Servers* that they represent. Security for these *Servers* is supported at two levels.

First, each OPC UA *Service* request contains a string parameter that is used to carry an audit record id. A *Client*, or any *Server* operating as a *Client*, such as an aggregating *Server*, can create a local audit log entry for a request that it submits. This parameter allows the *Client* to pass the identifier for this entry with the request. If the *Server* also maintains an audit log, then it can include this id in the audit log entry that it writes. When the log is examined and the entry is found, the examiner will be able to relate it directly to the audit log entry created by the *Client*. This capability allows for traceability across audit logs within a system. See IEC TR 62541-2 for additional information on auditing. A *Server* that maintains an audit log shall provide the information in the audit log entries via event *Messages* defined in this standard. The *Server* may choose to only provide the *Audit* information via event *Messages*. The *Audit EventType* is defined in IEC 62541-3.

Second, these aggregating *Servers* may open independent *Sessions* to the underlying *Servers* for each *Client* that accesses data from them. Figure 14 illustrates this concept.

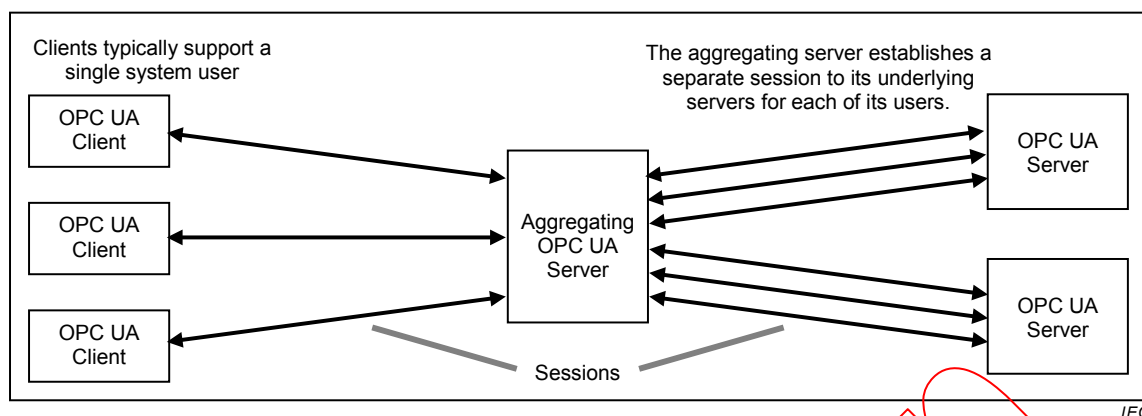


Figure 14 – Multiplexing Users on a Session

5.6.2.2 Parameters

Table 11 defines the parameters for the *Service*.

Table 11 – CreateSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The <i>authenticationToken</i> is always omitted. The type <i>RequestHeader</i> is defined in 7.26.
clientDescription	Application Description	Information that describes the <i>Client</i> application. The type <i>ApplicationDescription</i> is defined in 7.1.
serverUri	String	This value is only specified if the <i>EndpointDescription</i> has a <i>gatewayServerUri</i> . This value is the <i>applicationUri</i> from the <i>EndpointDescription</i> which is the <i>applicationUri</i> for the underlying <i>Server</i> . The type <i>EndpointDescription</i> is defined in 7.9.
endpointUrl	String	The network address that the <i>Client</i> used to access the <i>Session Endpoint</i> . The <i>HostName</i> portion of the URL should be one of the <i>HostNames</i> for the application that are specified in the <i>Server's ApplicationInstanceCertificate</i> (see 7.2). The <i>Server</i> shall raise an <i>AuditUriMismatchEventType</i> event if the URL does not match the <i>Server's HostNames</i> . <i>AuditUriMismatchEventType</i> event type is defined in IEC 62541-5. The <i>Server</i> uses this information for diagnostics and to determine the set of <i>EndpointDescriptions</i> to return in the response.
sessionName	String	Human readable string that identifies the <i>Session</i> . The <i>Server</i> makes this name and the <i>sessionId</i> visible in its <i>AddressSpace</i> for diagnostic purposes. The <i>Client</i> should provide a name that is unique for the instance of the <i>Client</i> . If this parameter is not specified the <i>Server</i> shall assign a value.
clientNonce	ByteString	A random number that should never be used in any other request. This number shall have a minimum length of 32 bytes. Profiles may increase the required length. The <i>Server</i> shall use this value to prove possession of its <i>Application Instance Certificate</i> in the response.
clientCertificate	ApplicationInstance Certificate	The <i>Application Instance Certificate</i> issued to the <i>Client</i> . The <i>ApplicationInstanceCertificate</i> type is defined in 7.2. If the <i>securityPolicyUri</i> is <i>None</i> , the <i>Server</i> shall ignore the <i>ApplicationInstanceCertificate</i> .
Requested SessionTimeout	Duration	Requested maximum number of milliseconds that a <i>Session</i> should remain open without activity. If the <i>Client</i> fails to issue a <i>Service</i> request within this interval, then the <i>Server</i> shall automatically terminate the <i>Client Session</i> .
maxResponse MessageSize	UInt32	The maximum size, in bytes, for the body of any response message. The <i>Server</i> should return a <i>Bad_ResponseTooLarge</i> service fault if a response message exceeds this limit. The value zero indicates that this parameter is not used. More information on the use of this parameter is provided in 5.3.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> type).
sessionId	NodeId	A unique <i>NodeId</i> assigned by the <i>Server</i> to the <i>Session</i> . This identifier is used to access the diagnostics information for the <i>Session</i> in the <i>Server</i> address space. It is also used in the audit logs and any events that report information related to the <i>Session</i> . The <i>Session</i> diagnostic information is described in IEC 62541-5.

Name	Type	Description
		and their related events are described in 6.2
authentication Token	Session AuthenticationToken	A unique identifier assigned by the <i>Server</i> to the <i>Session</i> . This identifier shall be passed in the <i>RequestHeader</i> of each request and is used with the <i>SecureChannelId</i> to determine whether a <i>Client</i> has access to the <i>Session</i> . This identifier shall not be reused in a way that the <i>Client</i> or the <i>Server</i> has a chance of confusing them with a previous or existing <i>Session</i> . The <i>SessionAuthenticationToken</i> type is described in 7.29.
revisedSession Timeout	Duration	Actual maximum number of milliseconds that a <i>Session</i> shall remain open without activity. The <i>Server</i> should attempt to honour the <i>Client</i> request for this parameter, but may negotiate this value up or down to meet its own constraints.
serverNonce	ByteString	A random number that should never be used in any other request. This number shall have a minimum length of 32 bytes. The <i>Client</i> shall use this value to prove possession of its <i>Application Instance Certificate</i> in the <i>ActivateSession</i> request. This value may also be used to prove possession of the <i>userIdentityToken</i> it specified in the <i>ActivateSession</i> request.
serverCertificate	ApplicationInstance Certificate	The <i>Application Instance Certificate</i> issued to the <i>Server</i> . A <i>Server</i> shall prove possession by using the private key to sign the <i>Nonce</i> provided by the <i>Client</i> in the request. The <i>Client</i> shall verify that this <i>Certificate</i> is the same as the one it used to create the <i>SecureChannel</i> . The <i>ApplicationInstanceCertificate</i> type is defined in 7.2. If the <i>securityPolicyUri</i> is NONE and none of the <i>UserTokenPolicies</i> requires encryption, the <i>Client</i> shall ignore the <i>ApplicationInstanceCertificate</i> .
serverEndpoints []	Endpoint Description	List of <i>Endpoints</i> that the server supports. The <i>Server</i> shall return a set of <i>EndpointDescriptions</i> available for the <i>serverUri</i> specified in the request. The <i>EndpointDescription</i> type is defined in 7.9. The <i>Client</i> shall verify this list with the list from a <i>Discovery Endpoint</i> if it used a <i>Discovery Endpoint</i> to fetch the <i>EndpointDescriptions</i> . It is recommended that <i>Servers</i> only include the <i>endpointUrl</i> , <i>securityMode</i> , <i>securityPolicyUri</i> , <i>userIdentityTokens</i> , <i>transportProfileUri</i> and <i>securityLevel</i> with all other parameters set to null. Only the recommended parameters shall be verified by the client.
serverSoftware Certificates []	SignedSoftware Certificate	This parameter is deprecated and the array shall be empty. The <i>SoftwareCertificates</i> are provided in the <i>Server AddressSpace</i> as defined in IEC 62541-5.
serverSignature	SignatureData	This is a signature generated with the private key associated with the <i>serverCertificate</i> . This parameter is calculated by appending the <i>clientNonce</i> to the <i>clientCertificate</i> and signing the resulting sequence of bytes. The <i>SignatureAlgorithm</i> shall be the <i>AsymmetricSignatureAlgorithm</i> specified in the <i>SecurityPolicy</i> for the <i>Endpoint</i> . The <i>SignatureData</i> type is defined in 7.30.
maxRequest MessageSize	UInt32	The maximum size, in bytes, for the body of any request message. The <i>Client Communication Stack</i> should return a <i>Bad_RequestTooLarge</i> error to the application if a request message exceeds this limit. The value zero indicates that this parameter is not used. 5.3 provides more information on the use of this parameter.

5.6.2.3 Service results

Table 12 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 12 – CreateSession Service Result Codes

Symbolic Id	Description
Bad_SecureChannelIdInvalid	See Table 165 for the description of this result code.
Bad_NonceInvalid	See Table 165 for the description of this result code. A server shall check the minimum length of the client nonce and return this status if the length is below 32 bytes. A check for duplicated nonces is optional and requires access to the nonce used to create the secure channel.
Bad_SecurityChecksFailed	See Table 165 for the description of this result code.
Bad_CertificateTimeInvalid	See Table 165 for the description of this result code.
Bad_CertificateIssuerTimeInvalid	See Table 165 for the description of this result code.
Bad_CertificateHostNameInvalid	See Table 165 for the description of this result code.
Bad_CertificateUriInvalid	See Table 165 for the description of this result code.
Bad_CertificateUseNotAllowed	See Table 165 for the description of this result code.
Bad_CertificateIssuerUseNotAllowed	See Table 165 for the description of this result code.
Bad_CertificateUntrusted	See Table 165 for the description of this result code.
Bad_CertificateRevocationUnknown	See Table 165 for the description of this result code.
Bad_CertificateIssuerRevocationUnknown	See Table 165 for the description of this result code.
Bad_CertificateRevoked	See Table 165 for the description of this result code.
Bad_CertificateIssuerRevoked	See Table 165 for the description of this result code.
Bad_TooManySessions	The server has reached its maximum number of sessions.
Bad_ServerUriInvalid	See Table 165 for the description of this result code.

5.6.3 ActivateSession

5.6.3.1 Description

This *Service* is used by the *Client* to submit its *SoftwareCertificates* to the *Server* for validation and to specify the identity of the user associated with the *Session*. This *Service* request shall be issued by the *Client* before it issues any other *Service* request after *CreateSession*. Failure to do so shall cause the *Server* to close the *Session*.

Whenever the *Client* calls this *Service* the *Client* shall prove that it is the same application that called the *CreateSession* *Service*. The *Client* does this by creating a signature with the private key associated with the *clientCertificate* specified in the *CreateSession* request. This signature is created by appending the last *serverNonce* provided by the *Server* to the *serverCertificate* and calculating the signature of the resulting sequence of bytes.

Once used, a *serverNonce* cannot be used again. For that reason, the *Server* returns a new *serverNonce* each time the *ActivateSession* *Service* is called.

When the *ActivateSession* *Service* is called for the first time then the *Server* shall reject the request if the *SecureChannel* is not same as the one associated with the *CreateSession* request. Subsequent calls to *ActivateSession* may be associated with different *SecureChannels*. If this is the case then the *Server* shall verify that the *Certificate* the *Client* used to create the new *SecureChannel* is the same as the *Certificate* used to create the original *SecureChannel*. In addition, the *Server* shall verify that the *Client* supplied a *UserIdentityToken* that is identical to the token currently associated with the *Session*. Once the *Server* accepts the new *SecureChannel* it shall reject requests sent via the old *SecureChannel*.

The *ActivateSession* *Service* is used to associate a user identity with a *Session*. When a *Client* provides a user identity then it shall provide proof that it is authorized to use that user identity. The exact mechanism used to provide this proof depends on the type of the *UserIdentityToken*. If the token is a *UserNameIdentityToken* then the proof is the *password* that is included in the token. If the token is an *X509IdentityToken* then the proof is a signature generated with private key associated with the *Certificate*. The data to sign is created by appending the last *serverNonce* to the *serverCertificate* specified in the *CreateSession* response. If a token includes a secret then it should be encrypted using the public key from the *serverCertificate*.

Clients can change the identity of a user associated with a *Session* by calling the *ActivateSession Service*. The *Server* validates the signatures provided with the request and then validates the new user identity. If no errors occur the *Server* replaces the user identity for the *Session*. Changing the user identity for a *Session* may cause discontinuities in active *Subscriptions* because the *Server* may have to tear down connections to an underlying system and re-establish them using the new credentials.

When a *Client* supplies a list of locale ids in the request, each locale id is required to contain the language component. It may optionally contain the <country/region> component. When the *Server* returns a *LocalizedText* in the context of the *Session*, it also may return both the language and the country/region or just the language as its default locale id.

When a *Server* returns a string to the *Client*, it first determines if there are available translations for it. If there are, then the *Server* returns the string whose locale id exactly matches the locale id with the highest priority in the *Client*-supplied list.

If there are no exact matches, then the *Server* ignores the <country/region> component of the locale id, and returns the string whose <language> component matches the <language> component of the locale id with the highest priority in the *Client* supplied list.

If there still are no matches, then the *Server* returns the string that it has along with the locale id.

A *Gateway Server* is expected to impersonate the user provided by the *Client* when it connects to the underlying *Server*. This means it shall re-calculate the signatures on the *UserIdentityToken* using the nonce provided by the underlying *Server*. The *Gateway Server* will have to use its own user credentials if the *UserIdentityToken* provided by the *Client* does not support impersonation.

5.6.3.2 Parameters

Table 13 defines the parameters for the *Service*.

Table 13 – ActivateSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The type <i>RequestHeader</i> is defined in 7.26.
clientSignature	SignatureData	This is a signature generated with the private key associated with the <i>clientCertificate</i> . The <i>SignatureAlgorithm</i> shall be the <i>AsymmetricSignatureAlgorithm</i> specified in the <i>SecurityPolicy</i> for the <i>Endpoint</i> . The <i>SignatureData</i> type is defined in 7.30.
clientSoftwareCertificates []	SignedSoftwareCertificate	These are the <i>SoftwareCertificates</i> which have been issued to the <i>Client</i> application. The <i>productUri</i> contained in the <i>SoftwareCertificates</i> shall match the <i>productUri</i> in the <i>ApplicationDescription</i> passed by the <i>Client</i> in the <i>CreateSession</i> requests. <i>Certificates</i> without matching <i>productUri</i> should be ignored. <i>Servers</i> may reject connections from <i>Clients</i> if they are not satisfied with the <i>SoftwareCertificates</i> provided by the <i>Client</i> . This parameter only needs to be specified in the first <i>ActivateSession</i> request after <i>CreateSession</i> . It shall always be omitted if the <i>maxRequestMessageSize</i> returned from the <i>Server</i> in the <i>CreateSession</i> response is less than one megabyte. The <i>SignedSoftwareCertificate</i> type is defined in 7.31.
localeIds []	LocaleId	List of locale ids in priority order for localized strings. The first <i>LocaleId</i> in the list has the highest priority. If the <i>Server</i> returns a localized string to the <i>Client</i> , the <i>Server</i> shall return the translation with the highest priority that it can. If it does not have a translation for any of the locales identified in this list, then it shall return the string value that it has and include the locale id with the string. See IEC 62541-3 for more detail on locale ids. If the <i>Client</i> fails to specify at least one locale id, the <i>Server</i> shall use any that it has. This parameter only needs to be specified during the first call to <i>ActivateSession</i> during a single application <i>Session</i> . If it is not specified the <i>Server</i> shall keep using the current <i>localeIds</i> for the <i>Session</i> .
userIdentityToken	Extensible Parameter UserIdentityToken	The credentials of the user associated with the <i>Client</i> application. The <i>Server</i> uses these credentials to determine whether the <i>Client</i> should be allowed to activate a <i>Session</i> and what resources the <i>Client</i> has access to during this <i>Session</i> . The <i>UserIdentityToken</i> is an extensible parameter type defined in 7.35. The <i>EndpointDescription</i> specifies what <i>UserIdentityTokens</i> the <i>Server</i> shall accept. Null or empty user token shall always be interpreted as anonymous.
userTokenSignature	SignatureData	If the <i>Client</i> specified a user identity token that supports digital signatures, then it shall create a signature and pass it as this parameter. Otherwise the parameter is omitted. The <i>SignatureAlgorithm</i> depends on the identity token type. The <i>SignatureData</i> type is defined in 7.30.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).
serverNonce	ByteString	A random number that should never be used in any other request. This number shall have a minimum length of 32 bytes. The <i>Client</i> shall use this value to prove possession of its <i>Application Instance Certificate</i> in the next call to <i>ActivateSession</i> request.
results []	StatusCode	List of validation results for the <i>SoftwareCertificates</i> (see 7.33 for <i>StatusCode</i> definition).
diagnosticInfos []	DiagnosticInfo	List of diagnostic information associated with <i>SoftwareCertificate</i> validation errors (see 7.8 for <i>DiagnosticInfo</i> definition). This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request.

5.6.3.3 Service results

Table 14 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 14 – ActivateSession Service Result Codes

Symbolic Id	Description
Bad_IdentityTokenInvalid	See Table 165 for the description of this result code.
Bad_IdentityTokenRejected	See Table 165 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_ApplicationSignatureInvalid	The signature provided by the client application is missing or invalid.
Bad_UserSignatureInvalid	The user token signature is missing or invalid.
Bad_NoValidCertificates	The <i>Client</i> did not provide at least one <i>Software Certificate</i> that is valid and meets the profile requirements for the <i>Server</i> .
Bad_IdentityChangeNotSupported	The <i>Server</i> does not support changing the user identity assigned to the session.

5.6.4 CloseSession

5.6.4.1 Description

This *Service* is used to terminate a *Session*. The *Server* takes the following actions when it receives a *CloseSession* request:

- It stops accepting requests for the *Session*. All subsequent requests received for the *Session* are discarded.
- It returns negative responses with the *StatusCode* *Bad_SessionClosed* to all requests that are currently outstanding to provide for the timely return of the *CloseSession* response. *Clients* are urged to wait for all outstanding requests to complete before submitting the *CloseSession* request.
- It removes the entry for the *Client* in its *SessionDiagnosticsArray Variable*.

5.6.4.2 Parameters

Table 15 defines the parameters for the *Service*.

Table 15 – CloseSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for <i>RequestHeader</i> definition).
deleteSubscriptions	Boolean	If the value is TRUE, the Server deletes all Subscriptions associated with the Session. If the value is FALSE, the Server keeps the Subscriptions associated with the Session until they timeout based on their own lifetime.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).

5.6.4.3 Service results

Table 16 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 16 – CloseSession Service Result Codes

Symbolic Id	Description
Bad_SessionIdInvalid	See Table 165 for the description of this result code.

5.6.5 Cancel

5.6.5.1 Description

This *Service* is used to cancel outstanding *Service* requests. Successfully cancelled service requests shall respond with *Bad_RequestCancelledByClient*.

5.6.5.2 Parameters

Table 17 defines the parameters for the *Service*.

Table 17 – Cancel Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for <i>RequestHeader</i> definition).
requestHandle	IntegerId	The <i>requestHandle</i> assigned to one or more requests that should be cancelled. All outstanding requests with the matching <i>requestHandle</i> shall be cancelled.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).
cancelCount	UInt32	Number of cancelled requests.

5.6.5.3 Service results

Common *StatusCodes* are defined in Table 165.

5.7 NodeManagement Service Set

5.7.1 Overview

This *Service Set* defines *Services* to add and delete *AddressSpace Nodes* and *References* between them. All added *Nodes* continue to exist in the *AddressSpace* even if the *Client* that created them disconnects from the *Server*.

5.7.2 AddNodes

5.7.2.1 Description

This *Service* is used to add one or more *Nodes* into the *AddressSpace* hierarchy. Using this *Service*, each *Node* is added as the *TargetNode* of a *HierarchicalReference* to ensure that the *AddressSpace* is fully connected and that the *Node* is added as a child within the *AddressSpace* hierarchy (see IEC 62541-3).

5.7.2.2 Parameters

Table 18 defines the parameters for the *Service*.

Table 18 – AddNodes Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for <i>RequestHeader</i> definition).
nodesToAdd []	AddNodesItem	List of <i>Nodes</i> to add. All <i>Nodes</i> are added as a <i>Reference</i> to an existing <i>Node</i> using a hierarchical <i>ReferenceType</i> . This structure is defined in-line with the following indented items.
parentNodeId	Expanded NodeId	<i>ExpandedNodeId</i> of the parent <i>Node</i> for the <i>Reference</i> . The <i>ExpandedNodeId</i> type is defined in 7.10.
referenceTypeId	NodeId	<i>NodeId</i> of the hierarchical <i>ReferenceType</i> to use for the <i>Reference</i> from the parent <i>Node</i> to the new <i>Node</i> .
requestedNewNodeId	Expanded NodeId	<i>Client</i> requested expanded <i>NodeId</i> of the <i>Node</i> to add. The <i>serverIndex</i> in the expanded <i>NodeId</i> shall be 0. If the <i>Server</i> cannot use this <i>NodeId</i> , it rejects this <i>Node</i> and returns the appropriate error code. If the <i>Client</i> does not want to request a <i>NodeId</i> , then it sets the value of this parameter to the null expanded <i>NodeId</i> . If the <i>Node</i> to add is a <i>ReferenceType Node</i> , its <i>NodeId</i> should be a numeric id. See IEC 62541-3 for a description of <i>ReferenceType NodeIds</i> .
browseName	QualifiedName	The browse name of the <i>Node</i> to add.
nodeClass	NodeClass	<i>NodeClass</i> of the <i>Node</i> to add.
nodeAttributes	Extensible Parameter NodeAttributes	The <i>Attributes</i> that are specific to the <i>NodeClass</i> . The <i>NodeAttributes</i> parameter type is an extensible parameter type specified in 7.18. A <i>Client</i> is allowed to omit values for some or all <i>Attributes</i> . If an <i>Attribute</i> value is omitted, the <i>Server</i> shall use the default values from the <i>TypeDefinitionNode</i> . If a <i>TypeDefinitionNode</i> was not provided the <i>Server</i> shall choose a suitable default value. The <i>Server</i> may still add an optional <i>Attribute</i> to the <i>Node</i> with an appropriate default value even if the <i>Client</i> does not specify a value.
typeDefinition	Expanded NodeId	<i>NodeId</i> of the <i>TypeDefinitionNode</i> for the <i>Node</i> to add. This parameter shall be null for all <i>NodeClasses</i> other than <i>Object</i> and <i>Variable</i> in which case it shall be provided.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).
results []	AddNodesResult	List of results for the <i>Nodes</i> to add. The size and order of the list matches the size and order of the <i>nodesToAdd</i> request parameter. This structure is defined in-line with the following indented items.
statusCode	StatusCode	<i>StatusCode</i> for the <i>Node</i> to add (see 7.33 for <i>StatusCode</i> definition).
addedNodeId	NodeId	<i>Server</i> assigned <i>NodeId</i> of the added <i>Node</i> . Null <i>NodeId</i> if the operation failed.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the <i>Nodes</i> to add (see 7.8 for <i>DiagnosticInfo</i> definition). The size and order of the list matches the size and order of the <i>nodesToAdd</i> request parameter. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request.

5.7.2.3 Service results

Table 19 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 19 – AddNodes Service Result Codes

Symbolic Id	Description
Bad_NothingToDo	See Table 165 for the description of this result code.
Bad_TooManyOperations	See Table 165 for the description of this result code.

5.7.2.4 StatusCodes

Table 20 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 20 – AddNodes Operation Level Result Codes

Symbolic Id	Description
Bad_ParentNodeInvalid	The parent node id does not refer to a valid node.
Bad_ReferenceTypeInvalid	See Table 166 for the description of this result code.
Bad_ReferenceNotAllowed	The reference could not be created because it violates constraints imposed by the data model.
Bad_NodeIdRejected	The requested node id was rejected either because it was invalid or because the server does not allow node ids to be specified by the client.
Bad_NodeIdExists	The requested node id is already used by another node.
Bad_NodeClassInvalid	See Table 166 for the description of this result code.
Bad_BrowseNameInvalid	See Table 166 for the description of this result code.
Bad_BrowseNameDuplicated	The browse name is not unique among nodes that share the same relationship with the parent.
Bad_NodeAttributesInvalid	The node <i>Attributes</i> are not valid for the node class.
Bad_TypeDefinitionInvalid	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.

5.7.3 AddReferences

5.7.3.1 Description

This *Service* is used to add one or more *References* to one or more *Nodes*. The *NodeClass* is an input parameter that is used to validate that the *Reference* to be added matches the *NodeClass* of the *TargetNode*. This parameter is not validated if the *Reference* refers to a *TargetNode* in a remote *Server*.

In certain cases, adding new *References* to the *AddressSpace* shall require that the *Server* add new *Server* ids to the *Server's ServerArray Variable*. For this reason, remote *Servers* are identified by their URI and not by their *ServerArray* index. This allows the *Server* to add the remote *Server* URIs to its *ServerArray*.

5.7.3.2 Parameters

Table 21 defines the parameters for the *Service*.

Table 21 – AddReferences Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for <i>RequestHeader</i> definition).
referencesToAdd []	AddReferences Item	List of <i>Reference</i> instances to add to the <i>SourceNode</i> . The <i>targetNodeClass</i> of each <i>Reference</i> in the list shall match the <i>NodeClass</i> of the <i>TargetNode</i> . This structure is defined in-line with the following indented items.
sourceNodeId	NodeId	<i>NodeId</i> of the <i>Node</i> to which the <i>Reference</i> is to be added. The source <i>Node</i> shall always exist in the <i>Server</i> to add the <i>Reference</i> . The <i>isForward</i> parameter can be set to FALSE if the target <i>Node</i> is on the local <i>Server</i> and the source <i>Node</i> on the remote <i>Server</i> .
referenceTypeId	NodeId	<i>NodeId</i> of the <i>ReferenceType</i> that defines the <i>Reference</i> .
isForward	Boolean	If the value is TRUE, the <i>Server</i> creates a forward <i>Reference</i> . If the value is FALSE, the <i>Server</i> creates an inverse <i>Reference</i> .
targetServerUri	String	URI of the remote <i>Server</i> . If this parameter is not null, it overrides the <i>serverIndex</i> in the <i>targetNodeId</i> .
targetNodeId	Expanded NodeId	Expanded <i>NodeId</i> of the <i>TargetNode</i> . The <i>ExpandedNodeId</i> type is defined in 7.10.
targetNodeClass	NodeClass	<i>NodeClass</i> of the <i>TargetNode</i> . The <i>Client</i> shall specify this since the <i>TargetNode</i> might not be accessible directly by the <i>Server</i> .
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).
results []	Status Code	List of <i>StatusCodes</i> for the <i>References</i> to add (see 7.33 for <i>StatusCode</i> definition). The size and order of the list matches the size and order of the <i>referencesToAdd</i> request parameter.
diagnosticInfos []	Diagnostic Info	List of diagnostic information for the <i>References</i> to add (see 7.8

Name	Type	Description
Request		
		<i>DiagnosticInfo</i> definition). The size and order of the list matches the size and order of the <i>referencesToAdd</i> request parameter. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request.

5.7.3.3 Service results

Table 22 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 22 – AddReferences Service Result Codes

Symbolic Id	Description
Bad_NothingToDo	See Table 165 for the description of this result code.
Bad_TooManyOperations	See Table 165 for the description of this result code.

5.7.3.4 StatusCodes

Table 23 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 23 – AddReferences Operation Level Result Codes

Symbolic Id	Description
Bad_SourceNodeIdInvalid	See Table 166 for the description of this result code.
Bad_ReferenceTypeIdInvalid	See Table 166 for the description of this result code.
Bad_ServerUriInvalid	See Table 165 for the description of this result code.
Bad_TargetNodeIdInvalid	See Table 166 for the description of this result code.
Bad_NodeClassInvalid	See Table 166 for the description of this result code.
Bad_ReferenceNotAllowed	The reference could not be created because it violates constraints imposed by the data model on this server.
Bad_ReferenceLocalOnly	The reference type is not valid for a reference to a remote <i>Server</i> .
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_DuplicateReferenceNotAllowed	The reference type between the nodes is already defined.
Bad_InvalidSelfReference	The server does not allow this type of self reference on this node.

5.7.4 DeleteNodes

5.7.4.1 Description

This *Service* is used to delete one or more *Nodes* from the *AddressSpace*.

When any of the *Nodes* deleted by an invocation of this *Service* is the *TargetNode* of a *Reference*, then those *References* are left unresolved based on the *deleteTargetReferences* parameter.

When any of the *Nodes* deleted by an invocation of this *Service* is being monitored, then a *Notification* containing the status code *Bad_NodeIdUnknown* is sent to the monitoring *Client* indicating that the *Node* has been deleted.

5.7.4.2 Parameters

Table 24 defines the parameters for the *Service*.

Table 24 – DeleteNodes Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for <i>RequestHeader</i> definition).
nodesToDelete []	DeleteNodes Item	List of <i>Nodes</i> to delete. This structure is defined in-line with the following indented items.
nodeId	NodeId	<i>NodeId</i> of the <i>Node</i> to delete.
deleteTargetReferences	Boolean	A <i>Boolean</i> parameter with the following values: TRUE delete <i>References</i> in <i>TargetNodes</i> that <i>Reference</i> the <i>Node</i> to delete. FALSE delete only the <i>References</i> for which the <i>Node</i> to delete is the source. The <i>Server</i> cannot guarantee that it is able to delete all <i>References</i> from <i>TargetNodes</i> if this parameter is TRUE.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for <i>ResponseHeader</i> definition).
results []	StatusCode	List of <i>StatusCodes</i> for the <i>Nodes</i> to delete (see 7.33 for <i>StatusCode</i> definition). The size and order of the list matches the size and order of the list of the <i>nodesToDelete</i> request parameter.
diagnosticInfos []	Diagnostic Info	List of diagnostic information for the <i>Nodes</i> to delete (see 7.8 for <i>DiagnosticInfo</i> definition). The size and order of the list matches the size and order of the <i>nodesToDelete</i> request parameter. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request.

5.7.4.3 Service results

Table 25 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 25 – DeleteNodes Service Result Codes

Symbolic Id	Description
Bad_NothingToDo	See Table 165 for the description of this result code.
Bad_TooManyOperations	See Table 165 for the description of this result code.

5.7.4.4 StatusCodes

Table 26 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 26 – DeleteNodes Operation Level Result Codes

Symbolic Id	Description
Bad_NodeIdInvalid	See Table 166 for the description of this result code.
Bad_NodeIdUnknown	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_NoDeleteRights	See Table 166 for the description of this result code.
Uncertain_ReferenceNotDeleted	The server was not able to delete all target references.

5.7.5 DeleteReferences

5.7.5.1 Description

This *Service* is used to delete one or more *References* of a *Node*.

When any of the *References* deleted by an invocation of this *Service* are contained in a *View*, then the *ViewVersion Property* is updated if this *Property* is supported.

Table 27 defines the parameters for the *Service*.