IEC 62541-4

Edition 1.0 2011-10

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 4: Services

Architecture unifiée OPC – Partie 4: Services

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2011 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 la CEI ou du Comité national de la CEI du pays du demandeur.
Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub

The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:
Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) 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 CEI

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

- Catalogue des publications de la CEI: www.iec.ch/searchpub/cur_fut-f.htm

Le Catalogue en-ligne de la CEI vous permet d'effectuer des recherches en utilisant différents critères (numéro de référence, texte, comité d'études,…). Il donne aussi des informations sur les projets et les publications retirées ou remplacées.

- Just Published CEI: www.iec.ch/online_news/justpub

Restez informé sur les nouvelles publications de la CEI. Just Published détaille deux fois par mois les nouvelles publications parues. Disponible en-ligne et aussi par email.

- Electropedia: www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 20 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International en ligne.

- Service Clients: www.iec.ch/webstore/custserv/custserv_entry-f.htm

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions, visitez le FAQ du Service clients ou contactez-nous:
Email: csc@iec.ch
Tél.: +41 22 919 02 11
Fax: +41 22 919 03 00

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

IEC 62541-4

Edition 1.0 2011-10

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 4: Services

Architecture unifiée OPC – Partie 4: Services

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XH
CODE PRIX

ICS 25.040.40; 25.100.01

ISBN 978-2-88912-730-6

® Registered trademark of the International Electrotechnical Commission
Marque déposée de la Commission Electrotechnique Internationale

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

CONTENTS

FOREWORD 12
INTRODUCTION 14
1 Scope 15
2 Normative references 15
3 Terms, definitions and conventions 16
3.1 Terms and definitions 16
3.2 Abbreviations 16
3.3 Conventions for Service definitions 17
4 Overview 18
4.1 Service Set model 18
4.2 Request/response Service procedures 21
5 Service Sets 21
5.1 General 21
5.2 Service request and response header 22
5.3 Service results 22
5.4 Discovery Service Set 23
5.4.1 Overview 23
5.4.2 FindServers 24
5.4.3 GetEndpoints 26
5.4.4 RegisterServer 29
5.5 SecureChannel Service Set 31
5.5.1 Overview 31
5.5.2 OpenSecureChannel 33
5.5.3 CloseSecureChannel 35
5.6 Session Service Set 35
5.6.1 Overview 35
5.6.2 CreateSession 36
5.6.3 ActivateSession 39
5.6.4 CloseSession 42
5.6.5 Cancel 42
5.7 NodeManagement Service Set 43
5.7.1 Overview 43
5.7.2 AddNodes 43
5.7.3 AddReferences 45
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 49
5.8.3 BrowseNext 51
5.8.4 TranslateBrowsePathsToNodeIds 53
5.8.5 RegisterNodes 55
5.8.6 UnregisterNodes 56
5.9 Query Service Set 56

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.9.1 Overview 56
5.9.2 Querying Views 57
5.9.3 QueryFirst 57
5.9.4 QueryNext 60
5.10 Attribute Service Set 61
5.10.1 Overview 61
5.10.2 Read 61
5.10.3 HistoryRead 63
5.10.4 Write 65
5.10.5 HistoryUpdate 67
5.11 Method Service Set 68
5.11.1 Overview 68
5.11.2 Call 68
5.12 MonitoredItem Service Set 70
5.12.1 MonitoredItem model 70
5.12.2 CreateMonitoredItems 74
5.12.3 ModifyMonitoredItems 76
5.12.4 SetMonitoringMode 78
5.12.5 SetTriggering 79
5.12.6 DeleteMonitoredItems 80
5.13 Subscription Service Set 81
5.13.1 Subscription model 81
5.13.2 CreateSubscription 88
5.13.3 ModifySubscription 89
5.13.4 SetPublishingMode 90
5.13.5 Publish 91
5.13.6 Republish 93
5.13.7 TransferSubscriptions 93
5.13.8 DeleteSubscriptions 95
6 Service behaviours 96
6.1 Security 96
6.1.1 Overview 96
6.1.2 Obtaining and Installing an Application Instance Certificate 96
6.1.3 Obtaining and installing a Software Certificate 97
6.1.4 Determining if a Certificate is Trusted 99
6.1.5 Validating a Software Certificate 101
6.1.6 Creating a SecureChannel 101
6.1.7 Creating a Session 103
6.1.8 Impersonating a User 104
6.2 Auditing 104
6.2.1 Overview 104
6.2.2 General audit logs 104
6.2.3 General audit Events 105
6.2.4 Auditing for Discovery Service Set 105
6.2.5 Auditing for SecureChannel Service Set 105
6.2.6 Auditing for Session Service Set 105
6.2.7 Auditing for NodeManagement Service Set 106
6.2.8 Auditing for Attribute Service Set 106
6.2.9 Auditing for Method Service Set 106

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.2.10 Auditing for View, Query, MonitoredItem and Subscription Service Set 107
6.3 Redundancy 107
6.3.1 Redundancy overview 107
6.3.2 Server redundancy overview 107
6.3.3 Client redundancy 109
7 Common parameter type definitions 110
7.1 ApplicationDescription 110
7.2 ApplicationInstanceCertificate 110
7.3 BrowseResult 111
7.4 ContentFilter 111
7.4.1 ContentFilter structure 111
7.4.2 ContentFilterResult 112
7.4.3 FilterOperator 113
7.4.4 FilterOperand parameters 119
7.5 Counter 120
7.6 ContinuationPoint 121
7.7 DataValue 121
7.7.1 General 121
7.7.2 PicoSeconds 121
7.7.3 SourceTimestamp 122
7.7.4 ServerTimestamp 122
7.7.5 *StatusCode* assigned to a value 123
7.8 DiagnosticInfo 123
7.9 EndpointDescription 124
7.10 ExpandedNodeId 124
7.11 ExtensibleParameter 125
7.12 Index 125
7.13 IntegerId 125
7.14 MessageSecurityMode 125
7.15 MonitoringParameters 125
7.16 MonitoringFilter parameters 126
7.16.1 Overview 126
7.16.2 DataChangeFilter 126
7.16.3 EventFilter 127
7.16.4 AggregateFilter 129
7.17 MonitoringMode 131
7.18 NodeAttributes parameters 131
7.18.1 Overview 131
7.18.2 ObjectAttributes parameter 132
7.18.3 VariableAttributes parameter 132
7.18.4 MethodAttributes parameter 133
7.18.5 ObjectTypeAttributes parameter 133
7.18.6 VariableTypeAttributes parameter 133
7.18.7 ReferenceTypeAttributes parameter 134
7.18.8 DataTypeAttributes parameter 134
7.18.9 ViewAttributes parameter 134
7.19 NotificationData parameters 135
7.19.1 Overview 135

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.19.2 DataChangeNotification parameter 135
7.19.3 EventNotificationList parameter 136
7.19.4 StatusChangeNotification parameter 136
7.20 NotificationMessage 136
7.21 NumericRange 137
7.22 QueryDataSet 138
7.23 ReadValueId 138
7.24 ReferenceDescription 139
7.25 RelativePath 139
7.26 RequestHeader 140
7.27 ResponseHeader 141
7.28 ServiceFault 141
7.29 SessionAuthenticationToken 141
7.30 SignatureData 143
7.31 SignedSoftwareCertificate 143
7.32 SoftwareCertificate 143
7.33 StatusCode 144
7.33.1 General 144
7.33.2 Common StatusCodes 146
7.34 TimestampsToReturn 148
7.35 UserIdentityToken parameters 148
7.35.1 Overview 148
7.35.2 AnonymousIdentityToken 149
7.35.3 UserNameIdentityToken 149
7.35.4 X509IdentityTokens 150
7.35.5 IssuedIdentityToken 150
7.36 UserTokenPolicy 151
7.37 ViewDescription 151
Annex A (informative) BNF definitions 152
Annex B (informative) Content Filter and Query Examples 154
Bibliography 172

Figure 1 – Discovery Service Set 18
Figure 2 – SecureChannel Service Set 18
Figure 3 – Session Service Set 19
Figure 4 – NodeManagement Service Set 19
Figure 5 – View Service Set 19
Figure 6 – Attribute Service Set 20
Figure 7 – Method Service Set 20
Figure 8 – MonitoredItem and Subscription Service Sets 21
Figure 9 – Discovery process 24
Figure 10 – Using a Gateway Server 28
Figure 11 – Registration process – Manually launched servers 29
Figure 12 – Registration process – Automatically Launched Servers 30
Figure 13 – SecureChannel and Session Services 32
Figure 14 – Multiplexing Users on a Session 37

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Figure 15 – MonitoredItem Model 70
Figure 16 – Typical delay in change detection 72
Figure 17 – Triggering Model 74
Figure 18 – Obtaining and installing an Application Instance Certificate 97
Figure 19 – Obtaining and Installing a Software Certificate 98
Figure 20 – Determining if a Application Instance Certificate is Trusted 101
Figure 21 – Establishing a SecureChannel 102
Figure 22 – Establishing a Session 103
Figure 23 – Impersonating a User 104
Figure 24 – Transparent Redundancy setup 108
Figure 25 – Non-Transparent Redundancy setup 108
Figure 26 – Redundancy mode 109
Figure 27 – Logical layers of a *Server* 142
Figure 28 – Obtaining a SessionAuthenticationToken 142
Figure B.1 – Filter Logic Tree Example 154
Figure B.2 – Filter Logic Tree Example 155
Figure B.3 – Example Type Nodes 157
Figure B.4 – Example Instance Nodes 158
Figure B.5 – Example 1 Filter 159
Figure B.6 – Example 2 Filter Logic Tree 160
Figure B.7 – Example 3 Filter Logic Tree 162
Figure B.8 – Example 4 Filter Logic Tree 164
Figure B.9 – Example 5 Filter Logic Tree 165
Figure B.10 – Example 6 Filter Logic Tree 166
The corresponding *ContentFilter* is illustrated in 168
Figure B.11 – Example 7 Filter Logic Tree 168
Figure B.12 – Example 8 Filter Logic Tree 169
Figure 13 – Example 9 Filter Logic Tree 171

Table 1 – Service Definition Table 17
Table 2 – Parameter Types defined in IEC 62541-3 17
Table 3 – FindServers Service Parameters 26
Table 4 – GetEndpoints Service Parameters 28
Table 5 – RegisterServer Service Parameters 31
Table 6 – RegisterServer Service Result Codes 31
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 39
Table 13 – ActivateSession Service Parameters 41
Table 14 – ActivateSession Service Result Codes 42

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 15 – CloseSession Service Parameters 42
Table 16 – CloseSession Service Result Codes 42
Table 17 – Cancel Service Parameters 43
Table 18 – AddNodes Service Parameters 44
Table 19 – AddNodes Service Result Codes 44
Table 20 – AddNodes Operation Level Result Codes 45
Table 21 – AddReferences Service Parameters 46
Table 22 – AddReferences Service Result Codes 46
Table 23 – AddReferences Operation Level Result Codes 46
Table 24 – DeleteNodes Service Parameters 47
Table 25 – DeleteNodes Service Result Codes 47
Table 26 – DeleteNodes Operation Level Result Codes 48
Table 27 – DeleteReferences Service Parameters 48
Table 28 – DeleteReferences Service Result Codes 49
Table 29 – DeleteReferences Operation Level Result Codes 49
Table 30 – Browse Service Parameters 50
Table 31 – Browse Service Result Codes 51
Table 32 – Browse Operation Level Result Codes 51
Table 33 – BrowseNext Service Parameters 52
Table 34 – BrowseNext Service Result Codes 52
Table 35 – BrowseNext Operation Level Result Codes 53
Table 36 – TranslateBrowsePathsToNodeIds Service Parameters 54
Table 37 – TranslateBrowsePathsToNodeIds Service Result Codes 54
Table 38 – TranslateBrowsePathsToNodeIds Operation Level Result Codes 55
Table 39 – RegisterNodes Service Parameters 55
Table 40 – RegisterNodes Service Result Codes 56
Table 41 – UnregisterNodes Service Parameters 56
Table 42 – UnregisterNodes Service Result Codes 56
Table 43 – QueryFirst Request Parameters 58
Table 44 – QueryFirst Response Parameters 59
Table 45 – QueryFirst Service Result Codes 60
Table 46 – QueryFirst Operation Level Result Codes 60
Table 47 – QueryNext Service Parameters 61
Table 48 – QueryNext Service Result Codes 61
Table 49 – Read Service Parameters 62
Table 50 – Read Service Result Codes 62
Table 51 – Read Operation Level Result Codes 63
Table 52 – HistoryRead ServiceParameters 63
Table 53 – HistoryRead Service Result Codes 64
Table 54 – HistoryRead Operation Level Result Codes 65
Table 55 – Write Service Parameters 66
Table 56 – Write Service Result Codes 66
Table 57 – Write Operation Level Result Codes 67

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 58 – HistoryUpdate Service Parameters 67
Table 59 – HistoryUpdate Service Result Codes 68
Table 60 – HistoryUpdate Operation Level Result Codes 68
Table 61 – Call Service Parameters 69
Table 62 – Call Service Result Codes 69
Table 63 – Call Operation Level Result Codes 70
Table 64 – CreateMonitoredItems Service Parameters 75
Table 65 – CreateMonitoredItems Service Result Codes 76
Table 66 – CreateMonitoredItems Operation Level Result Codes 76
Table 67 – ModifyMonitoredItems Service Parameters 77
Table 68 – ModifyMonitoredItems Service Result Codes 77
Table 69 – ModifyMonitoredItems Operation Level Result Codes 78
Table 70 – SetMonitoringMode Service Parameters 78
Table 71 – SetMonitoringMode Service Result Codes 78
Table 72 – SetMonitoringMode Operation Level Result Codes 79
Table 73 – SetTriggering Service Parameters 79
Table 74 – SetTriggering Service Result Codes 79
Table 75 – SetTriggering Operation Level Result Codes 80
Table 76 – DeleteMonitoredItems Service Parameters 80
Table 77 – DeleteMonitoredItems Service Result Codes 80
Table 78 – DeleteMonitoredItems Operation Level Result Codes 81
Table 79 – Subscription States 83
Table 80 – Subscription State Table 84
Table 81 – State variables and parameters 86
Table 82 – Functions 87
Table 83 – CreateSubscription Service Parameters 88
Table 84 – CreateSubscription Service Result Codes 89
Table 85 – ModifySubscription Service Parameters 89
Table 86 – ModifySubscription Service Result Codes 90
Table 87 – SetPublishingMode Service Parameters 90
Table 88 – SetPublishingMode Service Result Codes 90
Table 89 – SetPublishingMode Operation Level Result Codes 91
Table 90 – Publish Service Parameters 92
Table 91 – Publish Service Result Codes 92
Table 92 – Publish Operation Level Result Codes 92
Table 93 – Republish Service Parameters 93
Table 94 – Republish Service Result Codes 93
Table 95 – TransferSubscriptions Service Parameters 94
Table 96 – TransferSubscriptions Service Result Codes 94
Table 97 – TransferSubscriptions Operation Level Result Codes 94
Table 98 – DeleteSubscriptions Service Parameters 95
Table 99 – DeleteSubscriptions Service Result Codes 95
Table 100 – DeleteSubscriptions Operation Level Result Codes 95

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 101 – Certificate Validation Steps 100
Table 102 – Redundancy failover actions 109
Table 103 – ApplicationDescription 110
Table 104 – ApplicationInstanceCertificate 111
Table 105 – BrowseResult 111
Table 106 – ContentFilter Structure 112
Table 107 – ContentFilterResult Structure 112
Table 108 – ContentFilterResult Result Codes 112
Table 109 – ContentFilterResult Operand Result Codes 112
Table 110 – Basic FilterOperator Definition 113
Table 111 – Complex FilterOperator Definition 115
Table 112 – Wildcard characters 116
Table 113 – Conversion Rules 117
Table 114 – Data Precedence Rules 118
Table 115 – Logical AND Truth Table 118
Table 116 – Logical OR Truth Table 119
Table 117 – FilterOperand parameterTypeIds 119
Table 118 – ElementOperand 119
Table 119 – LiteralOperand 119
Table 120 – AttributeOperand 120
Table 121 – SimpleAttributeOperand 120
Table 122 – DataValue 121
Table 123 – DiagnosticInfo 123
Table 124 – EndpointDescription 124
Table 125 – ExpandedNodeId 124
Table 126 – ExtensibleParameter Base Type 125
Table 127 – MessageSecurityMode Values 125
Table 128 – MonitoringParameters 126
Table 129 – MonitoringFilter parameterTypeIds 126
Table 130 – DataChangeFilter 127
Table 131 – EventFilter structure 129
Table 132 – EventFilterResult structure 129
Table 133 – EventFilterResult Result Codes 129
Table 134 – AggregateFilter structure 130
Table 135 – AggregateFilterResult structure 130
Table 136 – MonitoringMode Values 131
Table 137 – NodeAttributes parameterTypeIds 131
Table 138 – Bit mask for specified Attributess 132
Table 139 – ObjectAttributes 132
Table 140 – VariableAttributes 133
Table 141 – MethodAttributes 133
Table 142 – ObjectTypeAttributes 133
Table 143 – VariableTypeAttributes 134

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 144 – ReferenceTypeAttributes 134
Table 145 – DataTypeAttributes 134
Table 146 – ViewAttributes 135
Table 147 – NotificationData parameterTypeIds 135
Table 148 – DataChangeNotification 136
Table 149 – EventNotificationList 136
Table 150 – StatusChangeNotification 136
Table 151 – NotificationMessage 137
Table 152 – NumericRange 138
Table 153 – QueryDataSet 138
Table 154 – ReadValueId 138
Table 155 – ReferenceDescription 139
Table 156 – RelativePath 139
Table 157 – RequestHeader 140
Table 158 – ResponseHeader 141
Table 159 – ServiceFault 141
Table 160 – SignatureData 143
Table 161 – SignedSoftwareCertificate 143
Table 162 – SoftwareCertificate 144
Table 163 – StatusCode Bit Assignments 145
Table 164 – DataValue InfoBits 145
Table 165 – Common Service Result Codes 146
Table 166 – Common Operation Level Result Codes 147
Table 167 – TimestampsToReturn Values 148
Table 168 – UserIdentityToken parameterTypeIds 148
Table 169 – UserIdentityToken Encrypted Token Format 149
Table 170 – AnonymousIdentityToken 149
Table 171 – UserNameIdentityToken 150
Table 172 – X509 Identity Token 150
Table 173 – IssuedIdentityToken 151
Table 174 – UserTokenPolicy 151
Table 175 – ViewDescription 151
Table A.1 – RelativePath 152
Table A.2 – RelativePath Examples 153
Table B.1 – ContentFilter Example 155
Table B.2 – ContentFilter Example 155
Table B.3 – Example 1 NodeTypeDescription 159
Table B.4 – Example 1 ContentFilter 159
Table B.5 – Example 1 QueryDataSets 159
Table B.6 – Example 2 NodeTypeDescription 160
Table B.7 – Example 2 ContentFilter 161
Table B.8 – Example 2 QueryDataSets 161
Table B.9 – Example 3 – NodeTypeDescriptions 161

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.10 – Example 3 ContentFilter 163
Table B.11 – Example 3 QueryDataSets 163
Table B.12 – Example 4 NodeTypeDescription 164
Table B.13 – Example 4 ContentFilter 164
Table B.14 – Example 4 QueryDataSets 164
Table B.15 – Example 5 NodeTypeDescription 165
Table B.16 – Example 5 ContentFilter 165
Table B.17 – Example 5 QueryDataSets 165
Table B.18 – Example 6 NodeTypeDescription 166
Table B.19 – Example 6 ContentFilter 166
Table B.20 – Example 6 QueryDataSets 167
Table B.21 – Example 6 QueryDataSets without Additional Information 167
Table B.22 – Example 7 NodeTypeDescription 168
Table B.23 – Example 7 ContentFilter 168
Table B.24 – Example 7 QueryDataSets 169
Table B.25 – Example 8 NodeTypeDescription 169
Table B.26 – Example 8 ContentFilter 170
Table B.27 – Example 8 QueryDataSets 170
Table B.28 – Example 9 NodeTypeDescription 170
Table B.29 – Example 9 ContentFilter 171
Table B.30 – Example 9 QueryDataSets 171

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

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.

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

FDIS	Report on voting
65E/191/FDIS	65E/213/RVD

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 document using a colour printer.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

INTRODUCTION

This International Standard is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 of IEC 62541. 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 of IEC 62541 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 referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC/TR 62541-1:2010, *OPC Unified Architecture – Part 1: Overview and Conce*pts

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*[1]

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*[2]

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*[3]

IEC 62541-8, *OPC Unified Architecture – Part 8: Data Access*[4]

ISO/IEC 7498 (all parts), *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*

http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

_____

1 To be published.

2 To be published.

3 To be published.

4 To be published.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

3 Terms, definitions and conventions

3.1 Terms and definitions

For the purposes of this document the following terms and definitions as well as the terms and definitions given in IEC/TR 62541-1, IEC/TR 62541-2 and IEC 62541-3 apply.

3.1.1
Deadband
permitted range for value changes that will not trigger a data change *Notification*

NOTE *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 Each *Server* may have multiple *Endpoints*. The address of an *Endpoint* must include a *HostName*.

3.1.3
Gateway Server
Server that acts as an intermediary for one or more *Servers*

NOTE *Gateway Servers* may be deployed to limit external access, provide protocol conversion or to provide features which the underlying *Servers* do not support.

3.1.4
HostName
unique identifier for a machine on a network

NOTE This identifier shall be unique within a local network, however, it may also be globally unique.

3.1.5
Security Token
identifier for a cryptographic key set

NOTE 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 Different installations of one software product could have the same software certificate.

3.2 Abbreviations

API Application Programming Interface
BNF Backus-Naur Form
CA Certificate Authority
CRL Certificate Revocation List
CTL Certificate Trust List
DA Data Access
UA Unified Architecture
URI Uniform Resource Identifier

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *Service*
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 *Service*

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

The term *Services* used in this standard is consistent with the definition provided in ISO/IEC 7498. The parameters of the Request and Indication service primitives are

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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*.

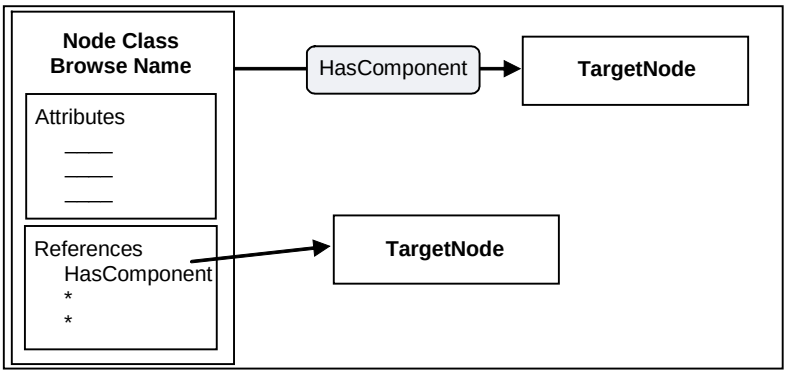


Figure 1 – Discovery Service Set

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*.

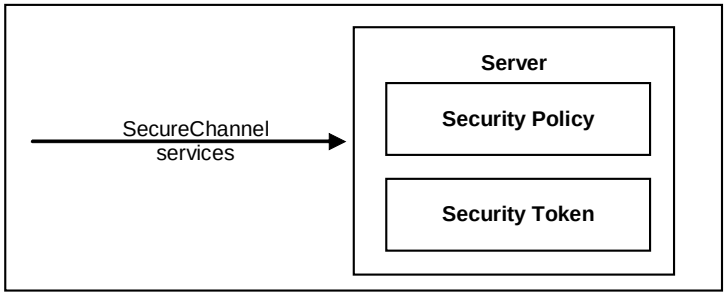


Figure 2 – SecureChannel Service Set

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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

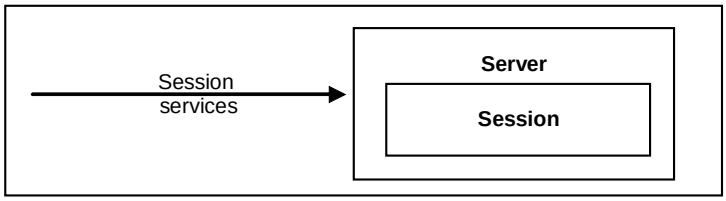


Figure 3 – Session Service Set

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

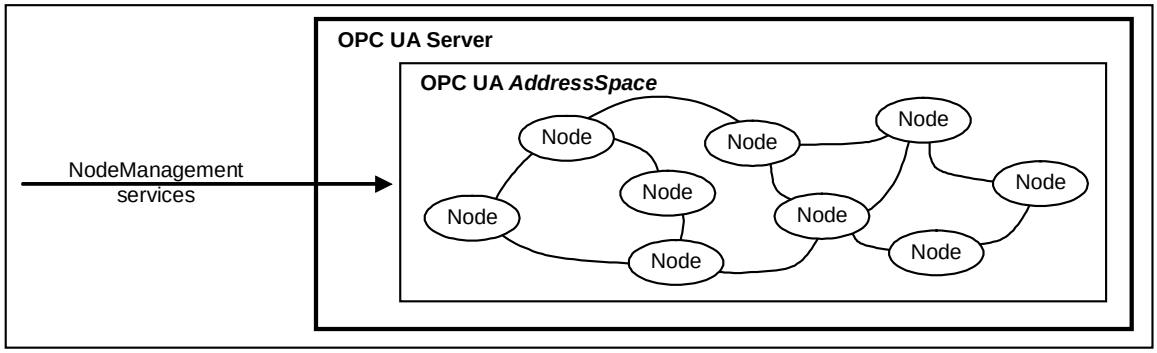


Figure 4 – NodeManagement Service Set

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*.

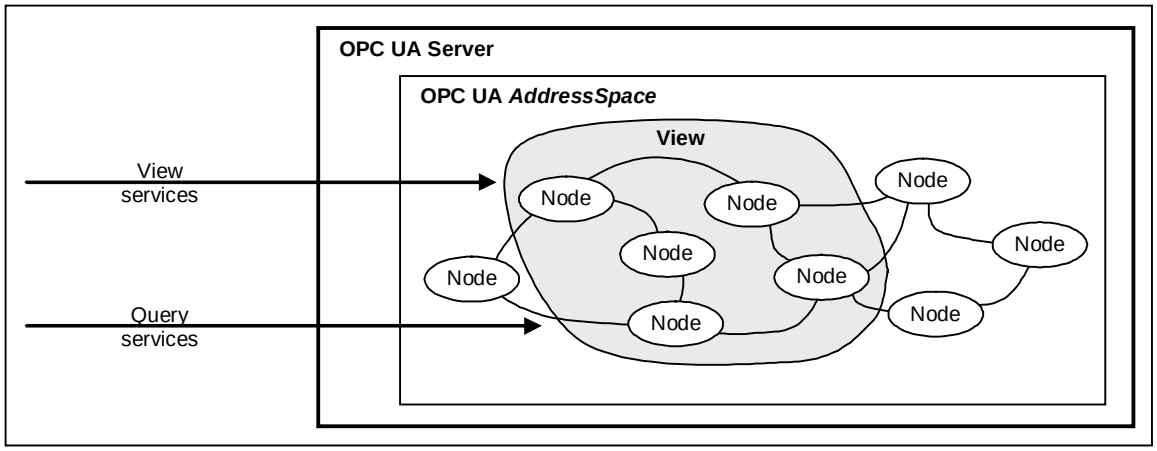


Figure 5 – View Service Set

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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

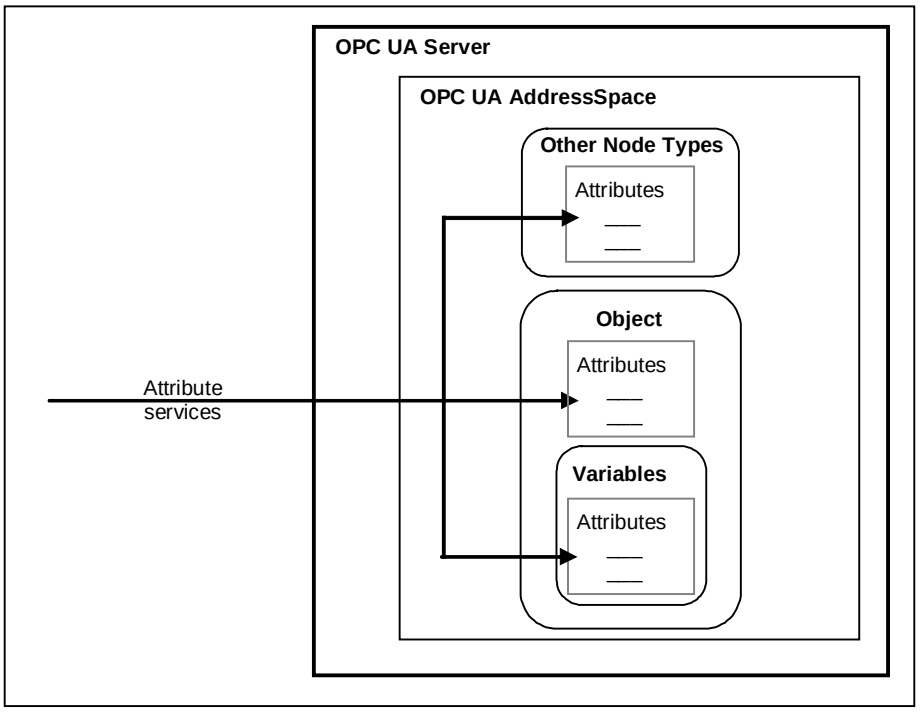


Figure 6 – Attribute Service Set

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.

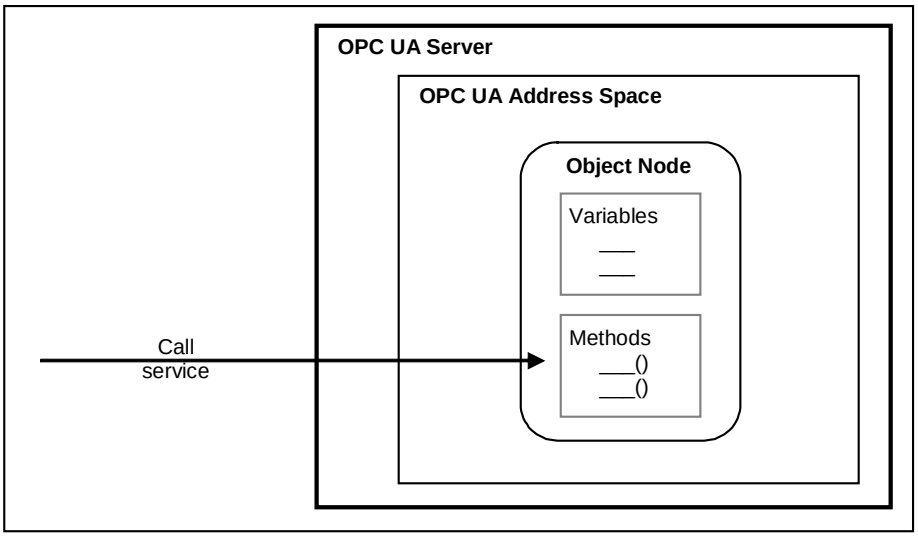


Figure 7 – Method Service Set

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 *Attribute*s 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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Subscription Services also provide for *Client* recovery from missed *Messages* and communication failures.

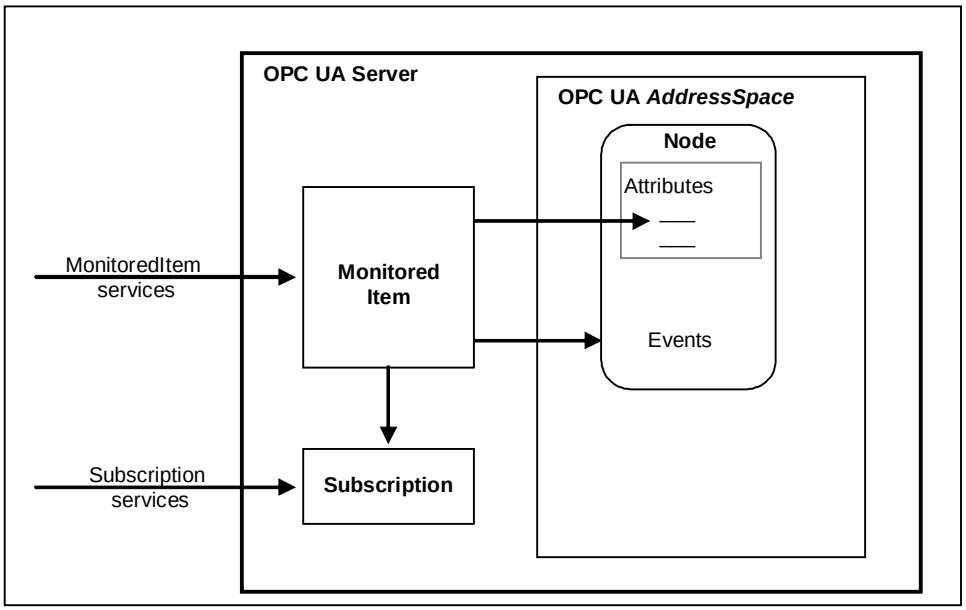


Figure 8 – MonitoredItem and Subscription Service Sets

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*. 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *StatusCode*s.

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 terminates 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 62541-12 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

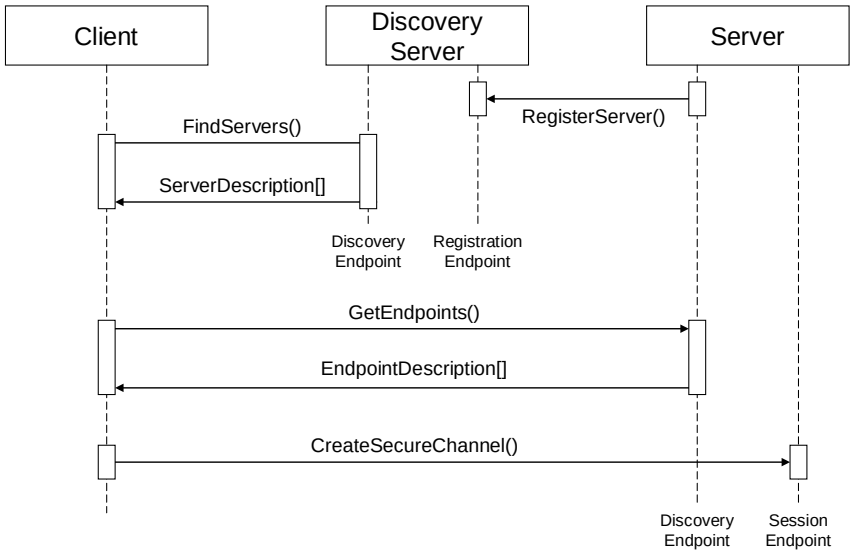


Figure 9 – Discovery process

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 C*lient* 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*.

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 *HostName* specified in the *Server Certificate* is the same as the *HostName* contained in the *endpointUrl* 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, the *Client* shall close the *SecureChannel* and report an error.

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 62541-12.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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*, however, the *Server* shall only return a single record that describes itself. *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 a an ordinary OPC UA *Server*.

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).

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.

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.

5.4.2.2 Parameters

Table 3 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 3 – FindServers Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The *authenticationToken* is always omitted. The type *RequestHeader* is defined in 7.26.
endpointUrl	String	The network address that the *Client* used to access the *Discovery Endpoint*. The *Server* uses this information for diagnostics and to determine what URLs to return in the response. The *Server* should return a suitable default URL if it does not recognize the *HostName* in the URL.
localeIds []	LocaleId	List of locales to use. The server should return the *ServerName* 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.
serverUris []	String	List of servers to return. All known servers are returned if the list is empty.
Response		
responseHeader	ResponseHeader	Common response parameters. The *ResponseHeader* type is defined in 7.27.
servers []	ApplicationDescription	List of *Servers* that meet the criteria specified in the request. This list is empty if no servers meet the criteria. The *ApplicationDescription* 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. 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 what type of user credentials shall be passed to the *Server* in the *ActivateSession* request (see 5.6.3).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

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 6.1.4) 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 *Server* via a *Gateway Server* is illustrated in Figure 10.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

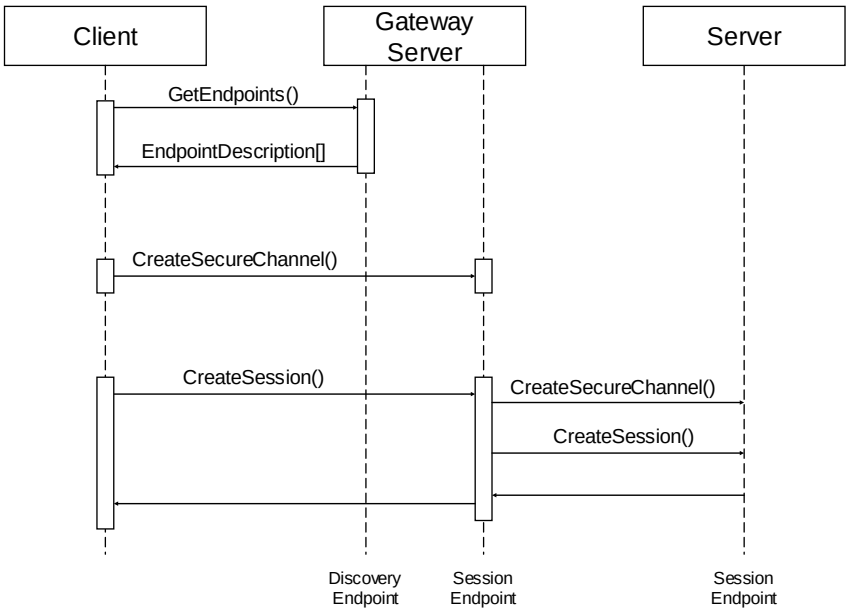


Figure 10 – Using a Gateway Server

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 *authenticationToken* is always omitted. The type *RequestHeader* is defined in 7.26.
endpointUrl	String	The network address that the *Client* used to access the *Discovery Endpoint*. The *Server* uses this information for diagnostics and to determine what URLs to return in the response. The *Server* should return a suitable default URL if it does not recognize the *HostName* 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 5.4.4.2.
profileUris []	String	List of profiles that the returned *Endpoints* shall support. All *Endpoints* are returned if the list is empty.
Response		
responseHeader	ResponseHeader	Common response parameters. The *ResponseHeader* type is defined in 7.27.
Endpoints []	EndpointDescription	List of *Endpoints* that meet the criteria specified in the request. This list is empty if no *Endpoints* meet the criteria. The *EndpointDescription* type is defined in 7.9.

5.4.3.3 Service Results

Common *StatusCodes* are defined in Table 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

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 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.

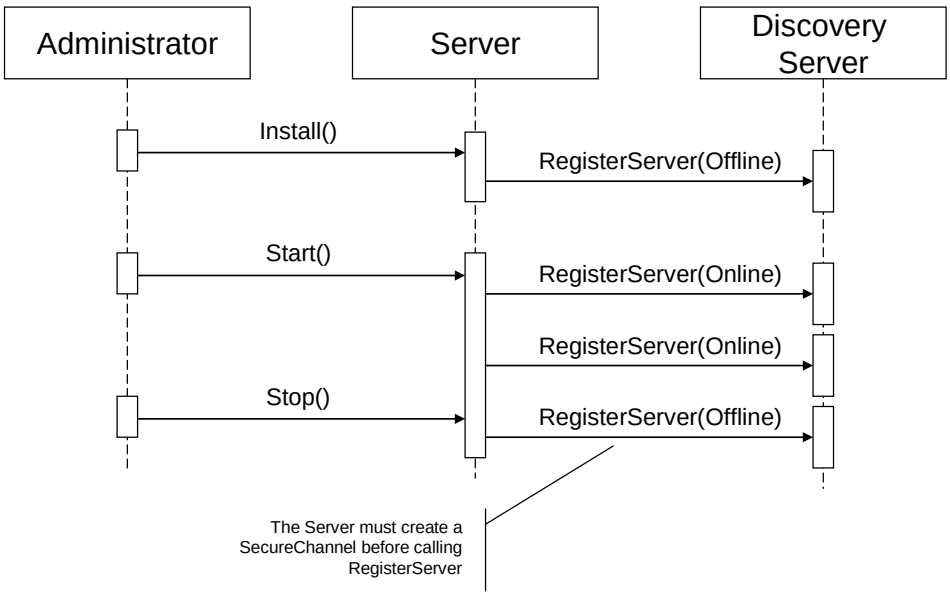


Figure 11 – Registration process – Manually launched servers

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

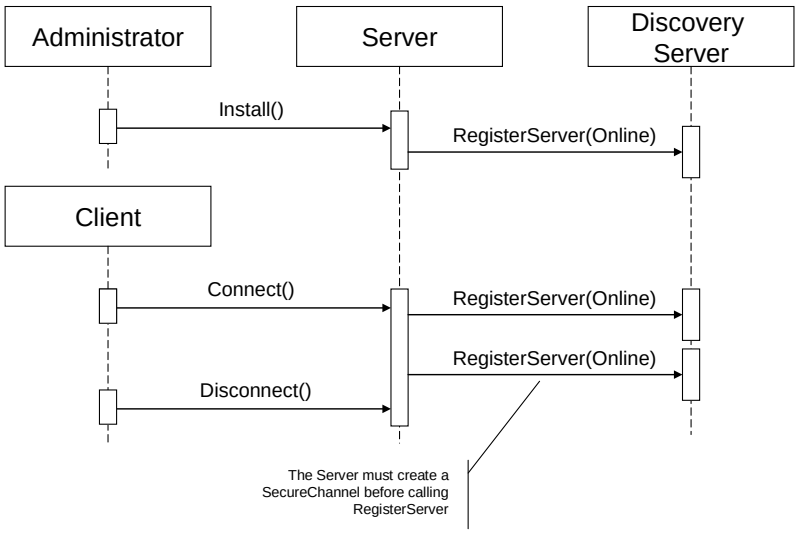


Figure 12 – Registration process – Automatically Launched Servers

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

Under normal conditions, *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 default is 10 min.

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 s but gradually increase until the registration frequency is the same as what it would be if no errors occurred. The recommended approach would double the period of each attempt until reaching the maximum.

When a *Server* registers with the a *Discovery Server* it may choose to provide 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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 5 – RegisterServer Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The *authenticationToken* is always omitted. The type *RequestHeader* is defined in 7.26.
server	RegisteredServer	The server to register.
serverUri	String	The globally unique identifier for the *Server* instance.
productUri	String	The globally unique identifier for the *Server* product.
serverNames []	LocalizedText	A list of localized descriptive names for the *Server*. The list shall have at least one valid entry.
serverType	Enum ApplicationType	The type of application. The enumeration values are defined in Table 103. The value “CLIENT_1” (The application is a *Client)* is not allowed.
gatewayServerUri	String	The URI of the *Gateway Server* associated with the *discoveryUrls*. This value is only specified by *Gateway Servers* that wish to register the *Servers* that they provide access to. For *Servers* that do not act as a *Gateway Server* this parameter shall be null.
discoveryUrls []	String	A list of *Discovery Endpoints* for the *Server*. The list shall have at least one valid entry.
semaphoreFilePath	String	The path to the semaphore file used to identify the server instance. The *Discovery Server* shall check that this file exists before returning the *ApplicationDescription* to the client. If the same semaphore file is used by another *Server* then that registration is deleted and replaced by the one being passed as part of this service invocation. If this value is null or empty then the *DiscoveryServer* does not attempt to verify the existence of the file.
isOnline	Boolean	True if the *Server* is currently able to accept connections from *Clients*.
Response		
ResponseHeader	ResponseHeader	Common response parameters. The type *ResponseHeader* 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_ServerUriInvalid	See Table 165 for the description of this result code.
Bad_ServerNameMissing	No *ServerName* was specified.
Bad_DiscoveryUrlMissing	No *DiscoveryUrl* was specified.
Bad_SempahoreFileMissing	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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

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*.

Note 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 a first step, the *SecureChannel Services* are used to establish a *SecureChannel* between the two *Communication Stacks* which allows the secure exchange of *Messages*. In a second step, the *OPC UA Applications* use the *Session Service Set* to establish a *OPC UA Application Session*.

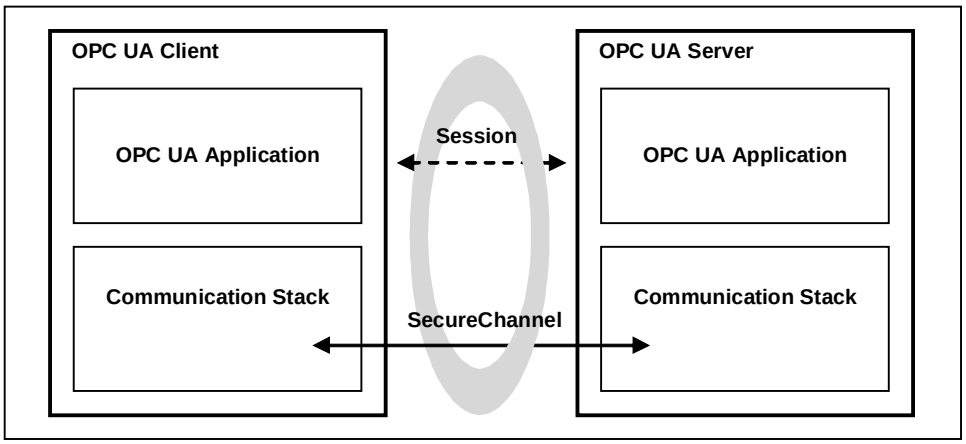


Figure 13 – SecureChannel and Session Services

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

impersonate the original *Client* by passing the original *Client's* user identity to the underlying *Server* when it calls the *ActiveSession 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* should 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.

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*.

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.

5.5.2.2 Parameters

Table 7 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 7 – OpenSecureChannel Service Parameters

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

5.5.2.3 Service results

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

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 *sessionId* is always omitted. The type *RequestHeader* is defined in 7.26.
secureChannelId	ByteString	The identifier for the *SecureChannel* to close.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* 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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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* address space. The second is the a*uthenticationToken* 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 a*uthenticationToken*. The *Server* may accept requests with the a*uthenticationToken* 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. Subclause 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 the *Application Instance Certificates* and the *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 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 match the *profileUri* specified in original the *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 response also contains a list of *SoftwareCertificates* that identify the capabilities of the *Server*. It 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 *SessionDiagnosticArray 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 described in IEC 62541-6.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *SessionDiagnosticArray 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, 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.

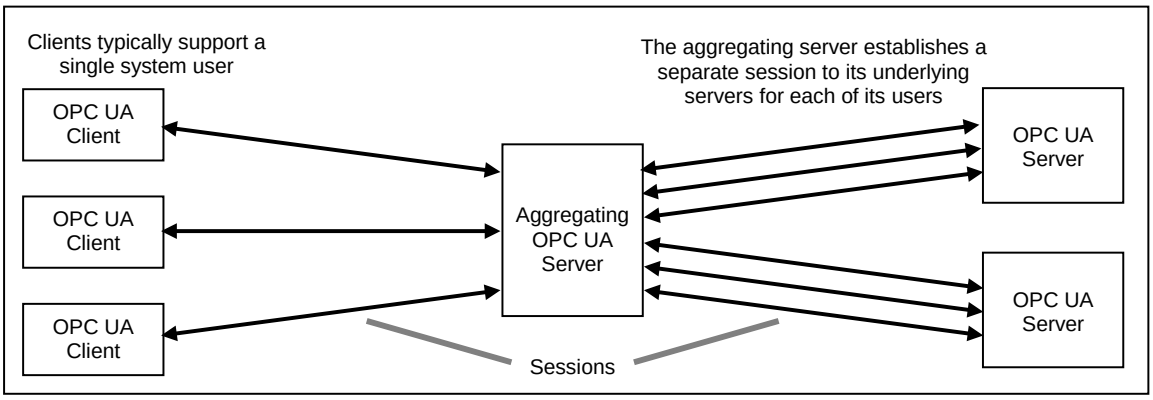


Figure 14 – Multiplexing Users on a Session

5.6.2.2 Parameters

Table 11 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 11 – CreateSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The *authenticationToken* is always omitted. The type *RequestHeader* is defined in 7.26.
clientDescription	Application Description	Information that describes the *Client* application. The type *ApplicationDescription* is defined in 7.1.
serverUri	String	This value is only specified if the *EndpointDescription* has a *gatewayServerUri*. This value is the *applicationUri* from the *EndpointDescription* which is the *applicationUri* for the underlying *Server*. The type *EndpointDescription* is defined in 7.9.
endpointUrl	String	The network address that the *Client* used to access the *Session Endpoint*. The *HostName* portion of the URL should be one of the *HostNames* for the application that are specified in the *Server's ApplicationInstanceCertificate* (see 7.2). The *Server* shall raise an *AuditUrlMismatchEventType* event if the URL does not match the Server's *HostNames*. *AuditUrlMismatchEventType* event type is defined in IEC 62541-5. The Server uses this information for diagnostics and to determine the set of *EndpointDescriptions* to return in the response.
sessionName	String	Human readable string that identifies the *Session*. The *Server* makes this name and the *sessionId* visible in its *AddressSpace* for diagnostic purposes. The *Client* should provide a name that is unique for the instance of the *Client*. If this parameter is not specified the *Server* 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 *Server* shall use this value to prove possession of its application instance *Certificate* in the response.
clientCertificate	ApplicationInstance Certificate	The application instance *Certificate* issued to the *Client*. The *ApplicationInstanceCertificate* type is defined in 7.2.
Requested SessionTimeout	Duration	Requested maximum number of milliseconds that a *Session* should remain open without activity. If the *Client* fails to issue a *Service* request within this interval, then the *Server* shall automatically terminate the *Client Session*.
maxResponse MessageSize	UInt32	The maximum size, in bytes, for the body of any response message. The *Server* should return a *Bad_ResponseTooLarge* service fault if a response message exceeds this limit. The value zero indicates that this parameter is not used. Subclause 5.3 provides more information on the use of this parameter.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* type).
sessionId	NodeId	A unique *NodeId* assigned by the *Server* to the *Session*. This identifier is used to access the diagnostics information for the *Session* in the *Server* address space. It is also used in the audit logs and any events that report information related to the *Session*. The *Session* diagnostic information is described in IEC 62541-5. Audit logs and their related events are described in 6.2
authentication Token	Session AuthenticationToken	A unique identifier assigned by the *Server* to the *Session*. This identifier shall be passed in the *RequestHeader* of each request and is used with the *SecureChannelId* to determine whether a *Client* has access to the *Session*. This identifier shall not be reused in a way that the *Client* or the *Server* has a chance of confusing them with a previous or existing *Session*. The *SessionAuthenticationToken* type is described in 7.29.
revisedSession Timeout	Duration	Actual maximum number of milliseconds that a *Session* shall remain open without activity. The *Server* should attempt to honour the *Client* 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 *Client* shall use this value to prove possession of its application instance *Certificate* in the *ActivateSession* request. This value may also be used to prove possession of the *userIdentityToken* it specified in the *ActivateSession* request.
serverCertificate	ApplicationInstance Certificate	The application instance *Certificate* issued to the *Server*. A *Server* shall prove possession by using the private key to sign the *Nonce* provided by the *Client* in the request. The *Client* shall verify that this *Certificate* is the same as the one it used to create the *SecureChannel*. The *ApplicationInstanceCertificate* type is defined in 7.2.
serverEndpoints []	Endpoint Description	List of *Endpoints* that the server supports. The *Server* shall return a set of *EndpointDescriptions* available for the *serverUri* specified in the request. The *EndpointDescription* type is defined in 7.9. The *Client* shall verify this list with the list from a *Discovery Endpoint* if it used a *Discovery Endpoint* to fetch the *EndpointDescriptions*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Name	Type	Description
serverSoftware Certificates []	SignedSoftware Certificate	These are the *SoftwareCertificates* which have been issued to the *Server* application. The *productUri* contained in the *SoftwareCertificates* shall match the *productUri* in the *EndpointDescription* used by the *Client* to connect to the *Server*. *Certificates* without matching *productUris* should be ignored. *Clients* should call *CloseSession* if they are not satisfied with the *SoftwareCertificates* provided by the *Server*. The *SignedSoftwareCertificate* type is defined in 7.31.
serverSignature	SignatureData	This is a signature generated with the private key associated with the *serverCertificate*. This parameter is calculated by appending the *clientNonce* to the *clientCertificate* and signing the resulting sequence of bytes. The *SignatureAlgorithm* shall be the *asymmetricSignature* algorithm specified in the *SecurityPolicy* for the *Endpoint*. The *SignatureData* type is defined in 7.30.
maxRequest MessageSize	UInt32	The maximum size, in bytes, for the body of any request message. The *Client Communication Stack* should return a *Bad_RequestTooLarge* error to the application if a request message exceeds this limit. The value zero indicates that this parameter is not used. Subclause 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.
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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

When the *ActivateSession Service* is called for the first time then the Server shall reject the request if the *SecureChannel* is not the 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 the 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 the response, 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, 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, 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, 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 nonces 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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 13 – ActivateSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters. The type *RequestHeader* is defined in 7.26.
clientSignature	SignatureData	This is a signature generated with the private key associated with the *clientCertificate*. The *SignatureAlgorithm* shall be the *asymmetricSignature* algorithm specified in the *SecurityPolicy* for the *Endpoint*. The *SignatureData* type is defined in 7.30.
clientSoftwareCertificates []	SignedSoftware Certificate	These are the *SoftwareCertificates* which have been issued to the *Client* application. The *productUri* contained in the *SoftwareCertificates* shall match the *productUri* in the Application*Description* passed by the *Client* in the *CreateSession* requests. *Certificates* without matching *productUris* should be ignored. *Servers* may reject connections from *Clients* if they are not satisfied with the *SoftwareCertificates* provided by the *Client*. This parameter only needs to be specified during the first call to *ActivateSession* for a single application *Session*. The *SignedSoftwareCertificate* type is defined in 7.31.
localeIds []	LocaleId	List of locale ids in priority order for localized strings. The first *localeId* in the list has the highest priority. If the *Server* returns a localized string to the *Client*, the *Server* 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 *Client* fails to specify at least one locale id, the *Server* shall use any that it has. This parameter only needs to be specified during the first call to *ActivateSession* during a single application *Session*. If it is not specified the *Server* shall keep using the current *localeIds* for the *Session*.
userIdentityToken	Extensible Parameter UserIdentityToken	The credentials of the user associated with the *Client* application. The *Server* uses these credentials to determine whether the *Client* should be allowed to activate a *Session* and what resources the *Client* has access to during this *Session*. The *UserIdentityToken* is an extensible parameter type defined in 7.35. The EndpointDescription specifies what *UserIdentityTokens* the Server shall accept.
userTokenSignature	SignatureData	If the *Client* 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 *SignatureAlgorithm* depends on the identity token type. The *SignatureData* type is defined in 7.30.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* 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 *Client* shall use this value to prove possession of its application instance *Certificate* in the next call to *ActivateSession* request.
results []	StatusCode	List of validation results for the *SoftwareCertificates* (see 7.33 for *StatusCode* definition).
diagnosticInfos []	DiagnosticInfo	List of diagnostic information associated with *SoftwareCertificate* validation errors (see 7.8 for *DiagnosticInfo* 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *Client* did not provide at least one software certificate that is valid and meets the profile requirements for the *Server*.
Bad_IdentityChangeNotSupported	The *Server* 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:

a) It stops accepting requests for the Session. All subsequent requests received for the Session are discarded.

b) 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.

c) It removes the entry for the *Client* in its *SessionDiagnosticArray 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 *RequestHeader* 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 *ResponseHeader* 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *RequestHeader* definition).
requestHandle	IntegerId	The *requestHandle* assigned to one or more requests that should be cancelled. All outstanding requests with the matching *requestHandles* shall be cancelled.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* 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*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 18 – AddNodes Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
nodesToAdd []	AddNodesItem	List of *Nodes* to add. All *Nodes* are added as a *Reference* to an existing *Node* using a hierarchical *ReferenceType*.
parentNodeId	Expanded NodeId	*ExpandedNodeId* of the parent *Node* for the *Reference*. The *ExpandedNodeId* type is defined in 7.10.
referenceTypeId	NodeId	*NodeId* of the hierarchical *ReferenceType* to use for the *Reference* from the parent *Node* to the new *Node*.
requestedNewNodeId	Expanded NodeId	*Client* requested expanded *NodeId* of the *Node* to add. The *serverIndex* in the expanded NodeId shall be 0. If the *Server* cannot use this *NodeId*, it rejects this *Node* and returns the appropriate error code. If the *Client* does not want to request a *NodeId*, then it sets the value of this parameter to the null expanded *NodeId*. If the *Node* to add is a *ReferenceType Node*, its *NodeId* should be a numeric id. See IEC 62541-3 for a description of *ReferenceType NodeIds*.
browseName	QualifiedName	The browse name of the *Node* to add.
nodeClass	NodeClass	*NodeClass* of the *Node* to add.
nodeAttributes	Extensible Parameter NodeAttributes	The *Attributes* that are specific to the *NodeClass*. The *NodeAttributes* parameter type is an extensible parameter type specified in 7.18. A *Client* is allowed to omit values for some or all *Attributes*. If an *Attribute* value is omitted, the *Server* shall use the default values from the *TypeDefinitionNode*. If a *TypeDefinitionNode* was not provided the *Server* shall choose a suitable default value. The *Server* may still add an optional Attribute to the *Node* with an appropriate default value even if the *Client* does not specify a value.
typeDefinition	Expanded NodeId	*NodeId* of the *TypeDefinitionNode* for the *Node* to add. This parameter shall be null for all *NodeClasses* other than *Object* and *Variable* in which case it shall be provided.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	AddNodesResult	List of results for the *Nodes* to add. The size and order of the list matches the size and order of the *nodesToAdd* request parameter.
statusCode	StatusCode	*StatusCode* for the *Node* to add (see 7.33 for *StatusCode* definition).
addedNodeId	NodeId	*Server* assigned *NodeId* of the added *Node*. Null *NodeId* if the operation failed.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *Nodes* to add (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *nodesToAdd* 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 20 – AddNodes Operation Level Result Codes

Symbolic Id	Description
Bad_ParentNodeIdInvalid	The parent node id does not refer to a valid node.
Bad_ReferenceTypeIdInvalid	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 *Attributes* 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 *ServerTable Variable*. For this reason, remote *Servers* are identified by their URI and not by their *ServerTable* index. This allows the *Server* to add the remote *Server* URIs to its *ServerTable*.

5.7.3.2 Parameters

Table 21 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 21 – AddReferences Service Parameters

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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 *RequestHeader* definition).
nodesToDelete []	DeleteNodes Item	List of *Nodes* to delete.
nodeId	NodeId	*NodeId* of the *Node* to delete.
deleteTargetReference	Boolean	A *Boolean* parameter with the following values: TRUE delete *References* in *TargetNodes* that *Reference* the *Node* to delete. FALSE delete only the *References* for which the *Node* to delete is the source. The *Server* cannot guarantee that he is able to delete all target *References* if this parameter is TRUE.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *Nodes* to delete (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the list of the *nodesToDelete* request parameter.
diagnosticInfos []	Diagnostic Info	List of diagnostic information for the *Nodes* to delete (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *nodesToDelete* 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

The deletion of a Reference shall trigger a *ModelChange Event*.

5.7.5.2 Parameters

Table 27 defines the parameters for the *Service*.

Table 27 – DeleteReferences Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
referencesToDelete []	DeleteReferences Item	List of *References* to delete.
sourceNodeId	NodeId	*NodeId* of the *Node* that contains the *Reference* to delete.
referenceTypeId	NodeId	*NodeId* of the *ReferenceType* that defines the *Reference* to delete.
isForward	Boolean	If the value is TRUE, the Server deletes a forward Reference. If the value is FALSE, the Server deletes an inverse Reference.
targetNodeId	ExpandedNodeId	*NodeId* of the *TargetNode* of the *Reference*. If the *Server* index indicates that the *TargetNode* is a remote *Node*, then the *nodeId* shall contain the absolute namespace URI. If the *TargetNode* is a local *Node* the *nodeId* shall contain the namespace index.
deleteBidirectional	Boolean	A *Boolean* parameter with the following values: TRUE delete the specified *Reference* and the opposite *Reference* from the *TargetNode*. If the *TargetNode* is located in a remote *Server*, the *Server* is permitted to delete the specified *Reference* only. FALSE delete only the specified *Reference*.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *References* to delete (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *referencesToDelete* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *References* to delete (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *referencesToDelete* 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.

5.7.5.3 Service results

Table 28 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 28 – DeleteReferences 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.

5.7.5.4 StatusCodes

Table 29 defines values for the r*esults* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 29 – DeleteReferences 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_ServerIndexInvalid	The server index is not valid.
Bad_TargetNodeIdInvalid	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.

5.8 View Service Set

5.8.1 Overview

Clients use the browse *Services* of the *View Service Set* to navigate through the *AddressSpace* or through a *View* which is a subset of the *AddressSpace*.

A *View* is a subset of the *AddressSpace* created by the *Server*. Future versions of this standard may also define services to create *Client*-defined *Views*. See IEC 62541-5 for a description of the organisation of views in the *AddressSpace*.

5.8.2 Browse

5.8.2.1 Description

This *Service* is used to discover the *References* of a specified *Node.* The browse can be further limited by the use of a *View*. This Browse *Service* also supports a primitive filtering capability.

5.8.2.2 Parameters

Table 30 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 30 – Browse Service Parameters

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>Request</td><td></td><td></td></tr>
<tr><td>requestHeader</td><td>RequestHeader</td><td>Common request parameters (see 7.26 for RequestHeader definition).</td></tr>
<tr><td>view</td><td>ViewDescription</td><td>Description of the View to browse (see 7.37 for ViewDescription definition). An empty ViewDescription value indicates the entire AddressSpace. Use of the empty ViewDescription value causes all References of the nodeToBrowse to be returned. Use of any other View causes only the References of the nodeToBrowse that are defined for that View to be returned.</td></tr>
<tr><td>requestedMax ReferencesPerNode</td><td>Counter</td><td>Indicates the maximum number of references to return for each starting Node specified in the request. The value 0 indicates that the Client is imposing no limitation (see 7.5 for Counter definition).</td></tr>
<tr><td>nodesToBrowse []</td><td>BrowseDescription</td><td>A list of nodes to Browse</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>NodeId of the Node to be browsed. The passed nodeToBrowse shall be part of the passed view.</td></tr>
<tr><td>browseDirection</td><td>enum BrowseDirection</td><td>An enumeration that specifies the direction of References to follow. It has the following values:
FORWARD_0 select only forward References.
INVERSE_1 select only inverse References.
BOTH_2 select forward and inverse References.
The returned References do indicate the direction the Server followed in the isForward parameter of the ReferenceDescription.
Symmetric References are always considered to be in forward direction therefore the isForward flag is always set to TRUE and symmetric References are not returned if browseDirection is set to INVERSE_1.</td></tr>
<tr><td>referenceTypeId</td><td>NodeId</td><td>Specifies the NodeId of the ReferenceType to follow. Only instances of this ReferenceType or its subtypes are returned.
If not specified then all References are returned.</td></tr>
<tr><td>includeSubtypes</td><td>Boolean</td><td>Indicates whether subtypes of the ReferenceType should be included in the browse. If TRUE, then instances of referenceTypeId and all of its subtypes are returned.</td></tr>
<tr><td>nodeClassMask</td><td>UInt32</td><td>Specifies the NodeClasses of the TargetNodes. Only TargetNodes with the selected NodeClasses are returned. The NodeClasses are assigned the following bits:
<table><tr><th>Bit</th><th>NodeClass</th></tr><tr><td>0</td><td>Object</td></tr><tr><td>1</td><td>Variable</td></tr><tr><td>2</td><td>Method</td></tr><tr><td>3</td><td>ObjectType</td></tr><tr><td>4</td><td>VariableType</td></tr><tr><td>5</td><td>ReferenceType</td></tr><tr><td>6</td><td>DataType</td></tr><tr><td>7</td><td>View</td></tr></table>If set to zero, then all NodeClasses are returned.</td></tr>
<tr><td>resultMask</td><td>UInt32</td><td>Specifies the fields in the ReferenceDescription structure that should be returned. The fields are assigned the following bits:
<table><tr><th>Bit</th><th>Result</th></tr><tr><td>0</td><td>ReferenceType</td></tr><tr><td>1</td><td>IsForward</td></tr><tr><td>2</td><td>NodeClass</td></tr><tr><td>3</td><td>BrowseName</td></tr><tr><td>4</td><td>DisplayName</td></tr><tr><td>5</td><td>TypeDefinition</td></tr></table>The ReferenceDescription type is defined in 7.24.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>Response</td><td></td><td></td></tr>
<tr><td>responseHeader</td><td>Response Header</td><td>Common response parameters (see 7.27 for ResponseHeader definition).</td></tr>
<tr><td>results []</td><td>BrowseResult</td><td>A list of BrowseResults. The size and order of the list matches the size and order of the nodesToBrowse specified in the request.
The BrowseResult type is defined in 7.3.</td></tr>
<tr><td>diagnosticInfos []</td><td>Diagnostic Info</td><td>List of diagnostic information for the results (see 7.8 for DiagnosticInfo definition). The size and order of the list matches the size and order of the results response 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.</td></tr>
</table>

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.8.2.3 Service results

Table 31 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 31 – Browse Service Result Codes

Symbolic Id	Description
Bad_ViewIdUnknown	The view id does not refer to a valid view *Node*.
Bad_ViewTimestampInvalid	See Table 165 for the description of this result code.
Bad_ViewParameterMismatchInvalid	See Table 165 for the description of this result code.
Bad_ViewVersionInvalid	See Table 165 for the description of this result code.
Bad_NothingToDo	See Table 165 for the description of this result code.
Bad_TooManyOperations	See Table 165 for the description of this result code.

5.8.2.4 StatusCodes

Table 32 defines values for the r*esults* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 32 – Browse 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_ReferenceTypeIdInvalid	See Table 166 for the description of this result code.
Bad_BrowseDirectionInvalid	See Table 166 for the description of this result code.
Bad_NodeNotInView	See Table 166 for the description of this result code.
Bad_NoContinuationPoints	See Table 166 for the description of this result code.
Uncertain_NotAllNodesAvailable	Browse results may be incomplete because of the unavailability of a subsystem.

5.8.3 BrowseNext

5.8.3.1 Description

This *Service* is used to request the next set of *Browse* or *BrowseNext* response information that is too large to be sent in a single response. “Too large” in this context means that the *Server* is not able to return a larger response or that the number of results to return exceeds the maximum number of results to return that was specified by the *Client* in the original Browse request. The *BrowseNext* shall be submitted on the same *Session* that was used to submit the Browse or *BrowseNext* that is being continued.

5.8.3.2 Parameters

Table 33 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 33 – BrowseNext Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
releaseContinuationPoints	Boolean	A *Boolean* parameter with the following values: TRUE passed *continuationPoints* shall be reset to free resources in the *Server*. FALSE passed *continuationPoints* shall be used to get the next set of browse information. A *Client* shall always use the continuation point returned by a *Browse* or *BrowseNext* response to free the resources for the continuation point in the *Server*. If the *Client* does not want to get the next set of browse information, *BrowseNext* shall be called with this parameter set to TRUE.
continuationPoints []	Continuation Point	A list of *Server*-defined opaque values that represent continuation points. The value for a continuation point was returned to the *Client* in a previous *Browse* or *BrowseNext* response. These values are used to identify the previously processed *Browse* or *BrowseNext* request that is being continued and the point in the result set from which the browse response is to continue. Clients may mix continuation points from different Browse or BrowseNext responses. The *ContinuationPoint* type is described in 7.6.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	BrowseResult	A list of references that met the criteria specified in the original *Browse* request. The size and order of this list matches the size and order of the *continuationPoints* request parameter. The *BrowseResult* type is defined in 7.3.
diagnosticInfos []	Diagnostic Info	List of diagnostic information for the *results* (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *results* response 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.

5.8.3.3 Service results

Table 34 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 34 – BrowseNext 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.

5.8.3.4 StatusCodes

Table 35 defines values for the r*esults* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 35 – BrowseNext 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_ReferenceTypeIdInvalid	See Table 166 for the description of this result code.
Bad_BrowseDirectionInvalid	See Table 166 for the description of this result code.
Bad_NodeNotInView	See Table 166 for the description of this result code.
Bad_ContinuationPointInvalid	See Table 166 for the description of this result code.

5.8.4 TranslateBrowsePathsToNodeIds

5.8.4.1 Description

This *Service* is used to request that the *Server* translates one or more browse paths to *NodeIds*. Each browse path is constructed of a starting *Node* and a *RelativePath*. The specified starting *Node* identifies the *Node* from which the *RelativePath* is based. The *RelativePath* contains a sequence of *ReferenceTypes* and *BrowseNames*.

One purpose of this *Service* is to allow programming against type definitions. Since *BrowseNames* shall be unique in the context of type definitions, a *Client* may create a browse path that is valid for a type definition and use this path on instances of the type. For example, an *ObjectType* “Boiler” may have a “HeatSensor” *Variable* as *InstanceDeclaration*. A graphical element programmed against the “Boiler” may need to display the *Value* of the “HeatSensor”. If the graphical element would be called “Boiler1”, an instance of “Boiler”, it would need to call this *Service* specifying the *NodeId* of “Boiler1” as starting *Node* and the *BrowseName* of the “HeatSensor” as browse path. The *Service* would return the *NodeId* of the “HeatSensor” of “Boiler1” and the graphical element could subscribe to its *Value Attribute*.

If a *Node* has multiple targets with the same *BrowseName*, the *Server* shall return a list of *NodeIds*. However, since one of the main purposes of this *Service* is to support programming against type definitions, the *NodeId* of the *Node* based on the type definition of the starting *Node* is returned as the first *NodeId* in the list.

5.8.4.2 Parameters

Table 36 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 36 – TranslateBrowsePathsToNodeIds Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
browsePaths []	BrowsePath	List of browse paths for which *NodeIds* are being requested.
startingNode	NodeId	*NodeId* of the starting *Node* for the browse path.
relativePath	RelativePath	The path to follow from the *startingNode*. The last element in the *relativePath* shall always have a *targetName* specified. This further restricts the definition of the RelativePath type. The *Server* shall return *Bad_BrowseNameInvalid* if the *targetName* is missing. The *RelativePath* structure is defined in 7.25.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	BrowsePathResult	List of results for the list of browse paths. The size and order of the list matches the size and order of the *browsePaths* request parameter.
statusCode	StatusCode	*StatusCode* for the browse path (see 7.33 for *StatusCode* definition).
targets []	BrowsePathTarget	List of targets for the *relativePath* from the *startingNode*. A *Server* may encounter a *Reference* to a *Node* in another *Server* which it can not follow while it is processing the *RelativePath*. If this happens the *Server* returns the *NodeId* of the external *Node* and sets the *remainingPathIndex* parameter to indicate which *RelativePath* elements still need to be processed. To complete the operation the *Client* shall connect to the other *Server* and call this service again using the target as the *startingNode* and the unprocessed elements as the *relativePath*.
targetId	ExpandedNodeId	The identifier for a target of the *RelativePath*.
remainingPathIndex	Index	The index of the first unprocessed element in the *RelativePath*. This value shall be equal to the maximum value of *Index* data type if all elements were processed (see 7.12 for *Index* definition).
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the list of browse paths (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *browsePaths* 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.

5.8.4.3 Service results

Table 37 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in 7.33.

Table 37 – TranslateBrowsePathsToNodeIds 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.

5.8.4.4 StatusCodes

Table 38 defines values for the operation level *statusCode* parameters that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 38 – TranslateBrowsePathsToNodeIds 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_NothingToDo	See Table 165 for the description of this result code. This code indicates that the relativePath contained an empty list.
Bad_BrowseNameInvalid	See Table 166 for the description of this result code. This code indicates that a TargetName was missing in a RelativePath.
Uncertain_ReferenceOutOfServer	The path element has targets which are in another server.
Bad_TooManyMatches	The requested operation has too many matches to return. Users should use queries for large result sets. *Servers* should allow at least 10 matches before returning this error code.
Bad_QueryTooComplex	The requested operation requires too many resources in the server.
Bad_NoMatch	The requested operation has no match to return.

5.8.5 RegisterNodes

5.8.5.1 Description

A *Server* often has no direct access to the information that it manages. Variables or services might be in underlying systems and additional effort is required to establish a connection to these systems. The *RegisterNodes Service* can be used by *Clients* to register the *Nodes* that they know they will access repeatedly (e.g. Write, Call). It allows *Servers* to set up anything needed so that the access operations will be more efficient. Clients can expect performance improvements when using registered *NodeIds*, but the optimization measures are vendor-specific. For *Variable Nodes Servers* shall concentrate their optimization efforts on the *Value Attribute*.

Registered *NodeIds* are only guaranteed to be valid within the current *Session*. *Clients* shall unregister unneeded Ids immediately to free up resources.

RegisterNodes does not validate the *NodeIds* from the request. *Servers* will simply copy unknown *NodeIds* in the response. Structural *NodeId* errors (size violations, invalid id types) will cause the complete *Service* to fail.

For the purpose of *Auditing*, *Servers* shall not use the registered *NodeIds*, but only the canonical *NodeId*s which is the value of the *NodeId Attribute*.

5.8.5.2 Parameters

Table 39 defines the parameters for the *Service*.

Table 39 – RegisterNodes Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
nodesToRegister []	NodeId	List of *NodeIds* to register that the client has retrieved through browsing, querying or in some other manner.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
registeredNodeIds []	NodeId	A list of *NodeIds* which the *Client* shall use for subsequent access operations. The size and order of this list matches the size and order of the *nodesToRegister* request parameter. The *Server* may return the *NodeId* from the request or a new (an alias) *NodeId*. It is recommended that the Server return a numeric *NodeIds* for aliasing. In case no optimization is supported for a *Node*, the *Server* shall return the *NodeId* from the request.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.8.5.3 Service results

Table 40 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 40 – RegisterNodes 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.
Bad_NodeIdInvalid	See Table 166 for the description of this result code. *Servers* shall completely reject the *RegisterNodes* request if any of the *NodeIds* in the *nodesToRegister* parameter are structurally invalid.

5.8.6 UnregisterNodes

5.8.6.1 Description

This *Service* is used to unregister *NodeIds* that have been obtained via the *RegisterNodes* service.

UnregisterNodes does not validate the *NodeIds* from the request. *Servers* shall simply unregister *NodeIds* that are known as registered *NodeIds*. Any *NodeIds* that are in the list, but are not registered *NodeIds* are simply ignored.

5.8.6.2 Parameters

Table 41 defines the parameters for the *Service*.

Table 41 – UnregisterNodes Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
nodesToUnregister []	NodeId	A list of *NodeIds* that have been obtained via the *RegisterNodes* service.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).

5.8.6.3 Service results

Table 42 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 42 – UnregisterNodes 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.

5.9 Query Service Set

5.9.1 Overview

This *Service Set* is used to issue a *Query* to a *Server*. OPC UA *Query* is generic in that it provides an underlying storage mechanism independent *Query* capability that can be used to access a wide variety of OPC UA data stores and information management systems. OPC UA *Query* permits a *Client* to access data maintained by a *Server* without any knowledge of the logical schema used for internal storage of the data. Knowledge of the *AddressSpace* is sufficient.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

An *OPC UA Application* is expected to use the OPC UA *Query Services* as part of an initialization process or an occasional information synchronization step. For example, OPC UA *Query* would be used for bulk data access of a persistent store to initialise an analysis application with the current state of a system configuration. A *Query* may also be used to initialise or populate data for a report.

A *Query* defines what instances of one or more *TypeDefinitionNodes* in the *AddressSpace* should supply a set of *Attributes*. Results returned by a *Server* are in the form of an array of *QueryDataSets*. The selected *Attribute* values in each *QueryDataSet* come from the definition of the selected *TypeDefinitionNodes or related TypeDefinitionNodes* and appear in results in the same order as the *Attributes* that were passed into the *Query*. *Query* also supports *Node* filtering on the basis of *Attribute* values, as well as relationships between *TypeDefinitionNodes*.

See Annex B for example queries.

5.9.2 Querying Views

A *View* is a subset of the *AddressSpace* available in the *Server*. See IEC 62541-5 for a description of the organisation of *Views* in the *AddressSpace*.

For any existing *View*, a *Query* may be used to return a subset of data from the *View*. When an application issues a *Query* against a *View*, only data defined by the *View* is returned. Data not included in the *View* but included in the original *AddressSpace* is not returned.

The *Query Services* supports access to current and historical data. The *Service* supports a *Client* querying a past version of the *AddressSpace.* Clients may specify a *ViewVersion* or a *Timestamp* in a *Query* to access past versions of the *AddressSpace*. OPC UA Query is complementary to Historical Access in that the former is used to *Query* an *AddressSpace* that existed at a time and the latter is used to *Query* for the value of *Attributes* over time. In this way, a *Query* can be used to retrieve a portion of a past *AddressSpace* so that *Attribute* value history may be accessed using Historical Access even if the *Node* is no longer in the current *AddressSpace*.

Servers that support *Query* are expected to be able to access the address space that is associated with the local *Server* and any *Views* that are available on the local *Server*. If a *View* or the address space also references a remote *Server*, query may be able to access the address space of remote *Server*, but it is not required. If a *Server* does access a remote *Server* the access shall be accomplished using the user identity of the *Client* as described in 5.5.1.

5.9.3 QueryFirst

5.9.3.1 Description

This *Service* is used to issue a *Query* request to the *Server*. The complexity of the *Query* can range from very simple to highly sophisticated. The *Query* can simply request data from instances of a *TypeDefinitionNode* or *TypeDefinitionNode* subject to restrictions specified by the filter. On the other hand, the *Query* can request data from instances of related *Node* types by specifying *a RelativePath* from an originating *TypeDefinitionNode.* In the filter, a separate set of paths can be constructed for limiting the instances that supply data. A filtering path can include multiple *RelatedTo* operators to define a multi-hop path between source instances and target instances. For example, one could filter on students that attend a particular school, but return information about students and their families. In this case, the student school relationship is traversed for filtering, but the student family relationship is traversed to select data. For a complete description of *ContentFilter*, see 7.4, also see Clause B.1 for simple examples and Clause B.2 for more complex examples of content filter and queries.

The *Client* provides an array of *NodeTypeDescription* which specify the *NodeId* of a *TypeDefinitionNode* and selects what *Attributes* are to be returned in the response. A client

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

can also provide a set of *RelativePaths* through the type system starting from an originating *TypeDefinitionNode*. Using these paths, the client selects a set of *Attributes* from *Nodes* that are related to instances of the originating *TypeDefinitionNode*. Additionally, the *Client* can request the *Server* return instances of subtypes of *TypeDefinitionNodes*. If a selected *Attribute* does not exist in a *TypeDefinitionNode* but does exist in a subtype, it is assumed to have a null value in the *TypeDefinitionNode* in question. Therefore, this does not constitute an error condition and a null value is returned for the *Attribute*.

The *Client* can use the filter parameter to limit the result set by restricting *Attributes* and *Properties* to certain values. Another way the *Client* can use a filter to limit the result set is by specifying how instances should be related, using *RelatedTo* operators. In this case, if an instance at the top of the *RelatedTo* path cannot be followed to the bottom of the path via specified hops, no *QueryDataSets* are returned for the starting instance or any of the intermediate instances.

When querying for related instances in the *RelativePath*, the *Client* can optionally ask for *References*. A *Reference* is requested via a RelativePath that only includes a *ReferenceType*. If all *References* are desired then the root *ReferenceType* would be listed. These *References* are returned as part of the *QueryDataSets*.

5.9.3.2 Parameters

Table 43 defines the request parameters and Table 44 the response parameters for the *QueryFirst Service*.

Table 43 – QueryFirst Request Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
view	ViewDescription	Specifies a *View* and temporal context to a *Server* (see 7.37 for *ViewDescription* definition).
nodeTypes []	NodeTypeDescription	This is the *Node* type description.
typeDefinitionNode	ExpandedNodeId	*NodeId* of the originating *TypeDefinitionNode* of the instances for which data is to be returned.
includeSubtypes	Boolean	A flag that indicates whether the *Server* should include instances of subtypes of the TypeDefinitionNode in the list of instances of the *Node* type.
dataToReturn []	QueryDataDescription	Specifies an *Attribute* or *Reference* from the originating typeDefinitionNode along a given relativePath for which to return data.
relativePath	RelativePath	Browse path relative to the originating Node that identifies the Node which contains the data that is being requested, where the originating *Node* is an instance *Node* of the type defined by the type definition *Node*. The instance *Nodes* are further limited by the filter provided as part of this call. For a definition of relativePath, see 7.25. This relative path could end on a *Reference*, in which case the *ReferenceDescription* of the *Reference* would be returned as its value.
attributeId	IntegerId	Id of the *Attribute*. This shall be a valid *Attribute* Id. The *IntegerId* is defined in 7.13. The IntegerIds for the Attributes are defined in IEC 62541-6. If the *RelativePath* ended in a *Reference* then this parameter is 0 and ignored by the server.
indexRange	NumericRange	This parameter is used to identify a single element of a structure or an array, or a single range of indexes for arrays. If a range of elements are specified, the values are returned as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is null if the specified *Attribute* is not an array or a structure. However, if the specified *Attribute* is an array or a structure, and this parameter is null, then all elements are to be included in the range.
filter	ContentFilter	Resulting *Nodes* shall be limited to the *Nodes* matching the criteria defined by the filter. ContentFilter is discussed in 7.4. If an empty filter is provided then the entire address space shall be examined and all *Nodes* that contain a matching requested *Attribute* or *Reference* are returned.
maxDataSetsToReturn	Counter	The number of *QueryDataSets* that the *Client* wants the *Server* to return in the response and on each subsequent continuation call

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Name	Type	Description
		response. The Server is allowed to further limit the response, but shall not exceed this limit. A value of 0 indicates that the *Client* is imposing no limitation.
maxReferencesToReturn	Counter	The number of *References* that the *Client* wants the *Server* to return in the response for each *QueryDataSet* and on each subsequent continuation call response. The Server is allowed to further limit the response, but shall not exceed this limit. A value of 0 indicates that the *Client* is imposing no limitation. For example a result where 4 *Nodes* are being returned, but each has 100 *References*, if this limit were set to 50 then only the first 50 *References* for each *Node* would be returned on the initial call and a continuation point would be set indicating additional data.

Table 44 – QueryFirst Response Parameters

Name	Type	Description
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
queryDataSets []	QueryDataSet	The array of *QueryDataSets*. This array is empty if no *Nodes* or *References* met the *nodeTypes* criteria. In this case the continuationPoint parameter shall be empty. The *QueryDataSet* type is defined in 7.22.
continuationPoint	ContinuationPoint	Server-defined opaque value that identifies the continuation point. The continuation point is used only when the *Query* results are too large to be returned in a single response. “Too large” in this context means that the *Server* is not able to return a larger response or that the number of *QueryDataSets* to return exceeds the maximum number of *QueryDataSets* to return that was specified by the *Client* in the request. The continuation point is used in the *QueryNext Service*. When not used, the value of this parameter is null. If a continuation point is returned, the *Client* shall call *QueryNext* to get the next set of *QueryDataSets* or to free the resources for the continuation point in the *Server*. A continuation point shall remain active until the *Client* passes the continuation point to *QueryNext* or the session is closed. If the maximum continuation points have been reached the oldest continuation point shall be reset. The *ContinuationPoint* type is described in 7.6.
parsingResults []	ParsingResult	List of parsing results for *QueryFirst*. The size and order of the list matches the size and order of the *NodeTypes* request parameter. This list is populated with any status codes that are related to the processing of the node types that are part of the query. The array can be empty if no errors where encountered. If any node type encountered an error all node types shall have an associated status code.
statusCode	StatusCode	Parsing result for the requested *NodeTypeDescription*.
dataStatusCodes []	StatusCode	List of results for *dataToReturn*. The size and order of the list matches the size and order of the *dataToReturn* request parameter. The array can be empty if no errors where encountered.
dataDiagnosticInfos []	DiagnosticInfo	List of diagnostic information *dataToReturn* (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *dataToReturn* 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 query request.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the requested *NodeTypeDescription*. 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 query request.
filterResult	ContentFilter Result	A structure that contains any errors associated with the filter. This structure shall be empty if no errors occurred. The *ContentFilterResult* type is defined in 7.4.2.

5.9.3.3 Service results

If the *Query* is invalid or cannot be processed, then *QueryDataSets* are not returned and only a *Service* result, filterResult, parsingResults and optional *DiagnosticInfo* is returned. Table 45 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 45 – QueryFirst 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.
Bad_ContentFilterInvalid	See Table 166 for the description of this result code.
Bad_ViewIdUnknown	See Table 165 for the description of this result code.
Bad_ViewTimestampInvalid	See Table 165 for the description of this result code.
Bad_ViewParameterMismatchInvalid	See Table 165 for the description of this result code.
Bad_ViewVersionInvalid	See Table 165 for the description of this result code.
Bad_InvalidFilter	The provided filter is invalid, see the filterResults for specific errors.
Bad_NodelistError	The NodeTypes provided contain an error, see parsingResults for specific errors.
Bad_InvalidView	The provided view description is not a valid View description.
Good_ResultsMayBeIncomplete	The server should have followed a reference to a node in a remote server but did not. The result set may be incomplete.

5.9.3.4 StatusCodes

Table 46 defines values for the parsingResults *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 46 – QueryFirst 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_NotTypeDefinition	The provided Nodeid was not a type definition nodeid.
Bad_AttribteIdInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.

5.9.4 QueryNext

5.9.4.1 Descriptions

This *Service* is used to request the next set of *QueryFirst* or *QueryNext* response information that is too large to be sent in a single response. “Too large” in this context means that the *Server* is not able to return a larger response or that the number of *QueryDataSets* to return exceeds the maximum number of *QueryDataSets* to return that was specified by the *Client* in the original request. The *QueryNext* shall be submitted on the same session that was used to submit the *QueryFirst* or *QueryNext* that is being continued.

5.9.4.2 Parameters

Table 47 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 47 – QueryNext Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
releaseContinuationPoint	Boolean	A *Boolean* parameter with the following values: TRUE passed *continuationPoint* shall be reset to free resources for the continuation point in the *Server*. FALSE passed *continuationPoint* shall be used to get the next set of *QueryDataSets*. A *Client* shall always use the continuation point returned by a *QueryFirst* or *QueryNext* response to free the resources for the continuation point in the *Server*. If the *Client* does not want to get the next set of *Query* information, *QueryNext* shall be called with this parameter set to TRUE. If the parameter is set to TRUE all array parameters in the response shall contain empty arrays.
continuationPoint	ContinuationPoint	Server defined opaque value that represents the continuation point. The value of the continuation point was returned to the *Client* in a previous *QueryFirst* or *QueryNext* response. This value is used to identify the previously processed *QueryFirst* or *QueryNext* request that is being continued, and the point in the result set from which the browse response is to continue. The *ContinuationPoint* type is described in 7.6.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
queryDataSets []	QueryDataSet	The array of *QueryDataSets*. The *QueryDataSet* type is defined in 7.22.
revisedContinuationPoint	ContinuationPoint	Server-defined opaque value that represents the continuation point. It is used only if the information to be returned is too large to be contained in a single response. When not used or when *releaseContinuationPoint* is set, the value of this parameter is null. The *ContinuationPoint* type is described in 7.6.

5.9.4.3 Service results

Table 48 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 48 – QueryNext Service Result Codes

Symbolic Id	Description
Bad_ContinuationPointInvalid	See Table 166 for the description of this result code.

5.10 Attribute Service Set

5.10.1 Overview

This *Service Set* provides *Services* to access *Attributes* that are part of *Nodes*.

5.10.2 Read

5.10.2.1 Description

This *Service* is used to read one or more *Attributes* of one or more *Nodes*. For constructed *Attribute* values whose elements are indexed, such as an array, this *Service* allows *Clients* to read the entire set of indexed values as a composite, to read individual elements or to read ranges of elements of the composite.

The maxAge parameter is used to direct the *Server* to access the value from the underlying data source, such as a device, if its copy of the data is older than that which the maxAge specifies. If the *Server* cannot meet the requested maximum age, it returns its "best effort" value rather than rejecting the request.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.10.2.2 Parameters

Table 49 defines the parameters for the *Service*.

Table 49 – Read Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
maxAge	Duration	Maximum age of the value to be read in milliseconds. The age of the value is based on the difference between the *ServerTimestamp* and the time when the *Server* starts processing the request. For example if the *Client* specifies a *maxAge* of 500 ms and it takes 100 ms until the *Server* starts processing the request, the age of the returned value could be 600 ms prior to the time it was requested. If the *Server* has one or more values of an *Attribute* that are within the maximum age, it can return any one of the values or it can read a new value from the data source. The number of values of an *Attribute* that a *Server* has depends on the number of *MonitoredItems* that are defined for the *Attribute*. In any case, the *Client* can make no assumption about which copy of the data will be returned. If the *Server* does not have a value that is within the maximum age, it shall attempt to read a new value from the data source. If the *Server* cannot meet the requested *maxAge*, it returns its "best effort" value rather than rejecting the request. This may occur when the time it takes the *Server* to process and return the new data value after it has been accessed is greater than the specified maximum age. If *maxAge* is set to 0, the *Server* shall attempt to read a new value from the data source. If *maxAge* is set to the max Int32 value or greater, the *Server* shall attempt to get a cached value. Negative values are invalid for *maxAge*.
timestampsTo Return	enum TimestampsTo Return	An enumeration that specifies the *Timestamps* to be returned for each requested *Variable Value Attribute*. The *TimestampsToReturn* enumeration is defined in 7.34.
nodesToRead []	ReadValueId	List of *Nodes* and their *Attributes* to read. For each entry in this list, a *StatusCode* is returned, and if it indicates success, the *Attribute Value* is also returned. The ReadValueId parameter type is defined in 7.23.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	DataValue	List of *Attribute* values (see 7.7 for *DataValue* definition). The size and order of this list matches the size and order of the *nodesToRead* request parameter. There is one entry in this list for each *Node* contained in the *nodesToRead* parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information (see 7.8 for *DiagnosticInfo* definition). The size and order of this list matches the size and order of the *nodesToRead* request parameter. There is one entry in this list for each *Node* contained in the *nodesToRead* 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.

5.10.2.3 Service results

Table 50 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 50 – Read 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.
Bad_MaxAgeInvalid	The max age parameter is invalid.
Bad_TimestampsToReturnInvalid	See Table 165 for the description of this result code.

5.10.2.4 StatusCodes

Table 51 defines values for the operation level *statusCode* contained in the *DataValue* structure of each *values* element. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 51 – Read 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_AttributeIdInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeNoData	See Table 166 for the description of this result code.
Bad_DataEncodingInvalid	See Table 166 for the description of this result code.
Bad_DataEncodingUnsupported	See Table 166 for the description of this result code.
Bad_NotReadable	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.

5.10.3 HistoryRead

5.10.3.1 Description

This *Service* is used to read historical values or *Events* of one or more *Nodes*. For constructed *Attribute* values whose elements are indexed, such as an array, this *Service* allows *Clients* to read the entire set of indexed values as a composite, to read individual elements or to read ranges of elements of the composite. *Servers* may make historical values available to *Clients* using this *Service*, although the historical values themselves are not visible in the *AddressSpace*.

The *AccessLevel Attribute* defined in IEC 62541-3 indicates a *Node*'s support for historical values. Several request parameters indicate how the *Server* shall access values from the underlying history data source. The *EventNotifier Attribute* defined in IEC 62541-3 indicates a *Node*'s support for historical *Events*.

The *continuationPoint* parameter in the *HistoryRead* is used to mark a point from which to continue the read if not all values could be returned in one response. The value is opaque for the *Client* and is only used to maintain the state information for the *Server* to continue from. A *Server* may use the timestamp of the last returned data item if the timestamp is unique. This can reduce the need in the *Server* to store state information for the continuation point.

For additional details on reading historical data and historical *Events* see IEC 62541-11.

5.10.3.2 Parameters

Table 52 defines the parameters for the *Service*.

Table 52 – HistoryRead ServiceParameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
historyReadDetails	Extensible Parameter HistoryReadDetails	The details define the types of history reads that can be performed. The *HistoryReadDetails* parameter type is an extensible parameter type formally defined in IEC 62541-11. The *ExtensibleParameter* type is defined in 7.11.
timestampsToReturn	enum TimestampsTo Return	An enumeration that specifies the timestamps to be returned for each requested *Variable Value Attribute*. The *TimestampsToReturn* enumeration is defined in 7.34. Specifying a *TimestampsToReturn* of NEITHER is not valid. A *Server* shall return a *Bad_InvalidTimestampArgument StatusCode* in this case.
releaseContinuation Points	Boolean	A *Boolean* parameter with the following values: TRUE passed *continuationPoints* shall be reset to free resources in the *Server*. FALSE passed *continuationPoints* shall be used to get the next set of historical information. A *Client* shall always use the continuation point returned by a *HistoryRead* response to free the resources for the continuation point in the *Server*. If the *Client* does not want to get the next set of historical information, *HistoryRead* shall be called with this parameter set to TRUE.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Name	Type	Description
Request		
nodesToRead []	HistoryReadValueId	This parameter contains the list of items upon which the historical retrieval is to be performed.
nodeId	NodeId	If the *HistoryReadDetails* is RAW, PROCESSED, MODIFIED or ATTIME: The *nodeId* of the *Nodes* whose historical values are to be read. The value returned shall always include a timestamp. If the *HistoryReadDetails* is EVENTS: The *NodeId* of the *Node* whose *Event* history is to be read. If the *Node* does not support the requested access for historical values or historical *Events* the appropriate error response for the given *Node* shall be generated.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. If a range of elements is specified, the values are returned as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is null if the value is not an array. However, if the value is an array, and this parameter is null, then all elements are to be included in the range.
dataEncoding	QualifiedName	A *QualifiedName* that specifies the data encoding to be returned for the *Value* to be read (see 7.23 for definition how to specify the data encoding). The parameter is ignored when reading history of *Events*.
continuationPoint	ByteString	For each *NodeToRead* this parameter specifies a continuation point returned from a previous *HistoryRead* call, allowing the *Client* to continue that read from the last value received. The *HistoryRead* is used to select an ordered sequence of historical values or events. A continuation point marks a point in that ordered sequence, such that the *Server* returns the subset of the sequence that follows that point. A null value indicates that this parameter is not used. This continuation point is described in more detail in IEC 62541-11.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* type).
results []	HistoryReadResult	List of read results. The size and order of the list matches the size and order of the *nodesToRead* request parameter.
statusCode	StatusCode	*StatusCode* for the *NodeToRead* (see 7.33 for *StatusCode* definition).
continuationPoint	ByteString	This parameter is used only if the number of values to be returned is too large to be returned in a single response. When this parameter is not used, its value is null. *Servers* shall support at least one continuation point per *Session*. *Servers* specify a max history continuation points per *Session* in the *Server* capabilities *Object* defined in IEC 62541-5. A continuation point shall remain active until the *Client* passes the continuation point to *HistoryRead* or the *Session* is closed. If the max continuation points have been reached the oldest continuation point shall be reset.
historyData	Extensible Parameter HistoryData	The history data returned for the *Node*. The *HistoryData* parameter type is an extensible parameter type formally defined in IEC 62541-11. It specifies the types of history data that can be returned. The *ExtensibleParameter* base type is defined in 7.11.
diagnosticInfos []	Diagnostic Info	List of diagnostic information. The size and order of the list matches the size and order of the *nodesToRead* request parameter. There is one entry in this list for each *Node* contained in the *nodesToRead* parameter. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in the processing of the request.

5.10.3.3 Service results

Table 53 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 53 – HistoryRead 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.
Bad_TimestampsToReturnInvalid	See Table 165 for the description of this result code.
Bad_HistoryOperationInvalid	See Table 166 for the description of this result code.
Bad_HistoryOperationUnsupported	See Table 166 for the description of this result code.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.10.3.4 StatusCodes

Table 54 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166. History access specific *StatusCodes* are defined in IEC 62541-11.

Table 54 – HistoryRead 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_DataEncodingInvalid	See Table 166 for the description of this result code.
Bad_DataEncodingUnsupported	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_ContinuationPointInvalid	See Table 165 for the description of this result code.
Bad_InvalidTimestampArgument	The defined timestamp to return was invalid.

5.10.4 Write

5.10.4.1 Description

This *Service* is used to write values to one or more *Attributes* of one or more *Nodes*. For constructed *Attribute* values whose elements are indexed, such as an array, this *Service* allows *Clients* to write the entire set of indexed values as a composite, to write individual elements or to write ranges of elements of the composite.

The values are written to the data source, such as a device, and the *Service* does not return until it writes the values or determines that the value cannot be written. In certain cases, the *Server* will successfully write to an intermediate system or *Server*, and will not know if the data source was updated properly. In these cases, the *Server* should report a success code that indicates that the write was not verified. In the cases where the *Server* is able to verify that it has successfully written to the data source, it reports an unconditional success.

It is possible that the *Server* may successfully write some *Attributes*, but not others. Rollback is the responsibility of the *Client*.

If a *Server* allows writing of *Attributes* with the *DataType* LocalizedText, the Client can add or overwrite the text for a locale by writing the text with the associated *LocaleId*. Writing a null *String* for a locale shall delete the *String* for that locale. Writing a null *String* for the text and a null *String* for the *LocaleId* shall delete the entries for all locales.

5.10.4.2 Parameters

Table 55 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 55 – Write Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
nodesToWrite []	WriteValue	List of *Nodes* and their *Attributes* to write.
nodeId	NodeId	*NodeId* of the *Node* that contains the *Attributes*.
attributeId	IntegerId	Id of the *Attribute*. This shall be a valid *Attribute* id. The *IntegerId* is defined in 7.13. The IntegerIds for the Attributes are defined in IEC 62541-6.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is not used if the specified *Attribute* is not an array. However, if the specified *Attribute* is an array and this parameter is not used, then all elements are to be included in the range. The parameter is null if not used.
value	DataValue	The *Node*'s *Attribute* value (see 7.7 for *DataValue* definition). If the *indexRange* parameter is specified then the *Value* shall be an array even if only one element is being written. If the *SourceTimestamp* or the *ServerTimestamp* is specified, the *Server* shall use these values. The *Server* returns a Bad_WriteNotSupported error if it does not support writing of timestamps. A Server shall return a Bad_TypeMismatch error if the data type of value is not the same as or a subtype of datatype for the *Attribute*. The *Server* returns a Bad_DataEncodingUnsupported error if it does not support writing of the passed data encoding.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of results for the *Nodes* to write (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *nodesToWrite* request parameter. There is one entry in this list for each *Node* contained in the *nodesToWrite* parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *Nodes* to write (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *nodesToWrite* 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.

5.10.4.3 Service results

Table 56 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 56 – Write 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.

5.10.4.4 StatusCodes

Table 57 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 57 – Write Operation Level Result Codes

Symbolic Id	Description
Good_CompletesAsynchronously	See Table 165 for the description of this result code. The value was successfully written to an intermediate system but the *Server* does not know if the data source was updated properly.
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_AttributeIdInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeNoData	See Table 166 for the description of this result code.
Bad_WriteNotSupported	The requested write operation is not supported. If a *Client* attempts to write any value, quality, timestamp combination and the *Server* does not support the requested combination (which could be a single quantity such as just timestamp), then the *Server* shall not perform any write on this *Node* and shall return this *StatusCode* for this *Node*.
Bad_NotWritable	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code. The current user does not have permission to write the attribute.
Bad_OutOfRange	See Table 166 for the description of this result code.
Bad_TypeMismatch	See Table 166 for the description of this result code.
Bad_DataEncodingUnsupported	See Table 166 for the description of this result code.

5.10.5 HistoryUpdate

5.10.5.1 Description

This *Service* is used to update historical values or *Events* of one or more *Nodes*. Several request parameters indicate how the *Server* is to update the historical value or *Event*. Valid actions are Insert, Replace or Delete.

5.10.5.2 Parameters

Table 58 defines the parameters for the *Service*.

Table 58 – HistoryUpdate Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
historyUpdateDetails []	Extensible Parameter HistoryUpdate Details	The details defined for this update. The *HistoryUpdateDetails* parameter type is an extensible parameter type formally defined in IEC 62541-11. It specifies the types of history updates that can be performed. The *ExtensibleParameter* type is defined in 7.11.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	HistoryUpdate Result	List of update results for the history update details. The size and order of the list matches the size and order of the details' element of the *historyUpdateDetails* parameter specified in the request.
statusCode	StatusCode	*StatusCode* for the update of the *Node* (see 7.33 for *StatusCode* definition).
operationResults []	StatusCode	List of *StatusCodes* for the operations to be performed on a *Node*. The size and order of the list matches the size and order of any list defined by the details' element being reported by this *updateResults* entry.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the operations to be performed on a *Node* (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of any list defined by the details' element being reported by this *updateResults* entry. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in the processing of the request.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the history update details. The size and order of the list matches the size and order of the details' element of the *historyUpdateDetails* parameter specified in the request. 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.10.5.3 Service results

Table 59 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 59 – HistoryUpdate 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.

5.10.5.4 StatusCodes

Table 60 defines values for the *statusCode* and *operationResult* parameters that are specific to this *Service*. Common *StatusCodes* are defined in Table 166. History access specific *StatusCodes* are defined in IEC 62541-11.

Table 60 – HistoryUpdate Operation Level Result Codes

Symbolic Id	Description
Bad_NotWritable	See Table 166 for the description of this result code.
Bad_HistoryOperationInvalid	See Table 166 for the description of this result code.
Bad_HistoryOperationUnsupported	See Table 166 for the description of this result code.

5.11 Method Service Set

5.11.1 Overview

Methods represent the function calls of *Objects*. They are defined in IEC 62541-3. *Methods* are invoked and return only after completion (successful or unsuccessful). Execution times for methods may vary, depending on the function that they perform.

The *Method Service Set* defines the means to invoke methods. A *method* shall be a *component* of an *Object*. Discovery is provided through the *Browse* and *Query Services*. *Clients* discover the *methods* supported by a *Server* by browsing for the owning *Objects References* that identify their supported *methods*.

Because *Methods* may control some aspect of plant operations, method invocation may depend on environmental or other conditions. This may be especially true when attempting to re-invoke a method immediately after it has completed execution. Conditions that are required to invoke the method might not yet have returned to the state that permits the method to start again.

5.11.2 Call

5.11.2.1 Description

This *Service* is used to call (invoke) a list of *Methods*. Each *method* call is invoked within the context of an existing *Session*. If the *Session* is terminated, the results of the *method's* execution cannot be returned to the *Client* and are discarded. This is independent of the task actually performed at the *Server*.

This *Service* provides for passing input and output arguments to/from a method. These arguments are defined by *Properties* of the method.

5.11.2.2 Parameters

Table 61 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 61 – Call Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
methodsToCall []	CallMetthodRequest	List of *Methods* to call.
objectId	NodeId	The *NodeId* shall be that of the *Object* or *ObjectType* that is the source of a *HasComponent Reference* (or subtype of *HasComponent Reference*) to this *Method*. See IEC 62541-3 for a description of *Objects* and their *Methods*.
methodId	NodeId	*NodeId* of the *Method* to invoke.
inputArguments []	BaseDataType	List of input argument values. An empty list indicates that there are no input arguments. The size and order of this list matches the size and order of the input arguments defined by the input *InputArguments Property* of the *Method*. The name, a description and the data type of each argument are defined by the *Argument* structure in each element of the method's *InputArguments Property*.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
result []	CallMethodResult	Result for the *Method* calls.
statusCode	StatusCode	*StatusCode* of the *Method* executed in the server. This *StatusCode* is set to the Bad_InvalidArgument *StatusCode* if at least one input argument broke a constraint (e.g. wrong data type, value out of range). This *StatusCode* is set to bad *StatusCode* if the *Method* execution failed in the server, e.g. based on an exception or an HRESULT.
inputArgumentResults []	StatusCode	List of *StatusCodes* for each *inputArgument*.
inputArgumentDiagnosticInfos []	DiagnosticInfo	List of diagnostic information for each *inputArgument*. 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.
outputArguments []	BaseDataType	List of output argument values. An empty list indicates that there are no output arguments. The size and order of this list matches the size and order of the output arguments defined by the *OutputArguments Property* of the *Method*. The name, a description and the data type of each argument are defined by the *Argument* structure in each element of the methods *OutputArguments Property*.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *StatusCode* of the *callResult*. 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.

5.11.2.3 Service results

Table 62 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 62 – Call 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.

5.11.2.4 StatusCodes

Table 63 defines values for the *statusCode* and *operationResult* parameters that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 63 – Call Operation Level Result Codes

Symbolic Id	Description
Bad_NodeIdInvalid	See Table 166 for the description of this result code. Used to indicate that the specified object is not valid.
Bad_NodeIdUnknown	See Table 166 for the description of this result code. Used to indicate that the specified object is not valid.
Bad_ArgumentsMissing	The client did not specify all of the input arguments for the method.
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_MethodInvalid	The method id does not refer to a method for the specified object.
Bad_OutOfRange	See Table 166 for the description of this result code. Used to indicate that an input argument is outside the acceptable range.
Bad_TypeMismatch	See Table 166 for the description of this result code. Used to indicate that an input argument does not have the correct data type.

5.12 MonitoredItem Service Set

5.12.1 MonitoredItem model

5.12.1.1 Overview

Clients define *MonitoredItems* to subscribe to data and *Events*. Each *MonitoredItem* identifies the item to be monitored and the *Subscription* to use to send *Notifications*. The item to be monitored may be any *Node Attribute*.

Notifications are data structures that describe the occurrence of data changes and *Events*. They are packaged into *NotificationMessages* for transfer to the *Client*. The *Subscription* periodically sends *NotificationMessages* at a user-specified publishing interval, and the cycle during which these messages are sent is called a publishing cycle.

Four primary parameters are defined for *MonitoredItems* that tell the *Server* how the item is to be sampled, evaluated and reported. These parameters are the sampling interval, the monitoring mode, the filter and the queue parameter. Figure 15 illustrates these concepts.

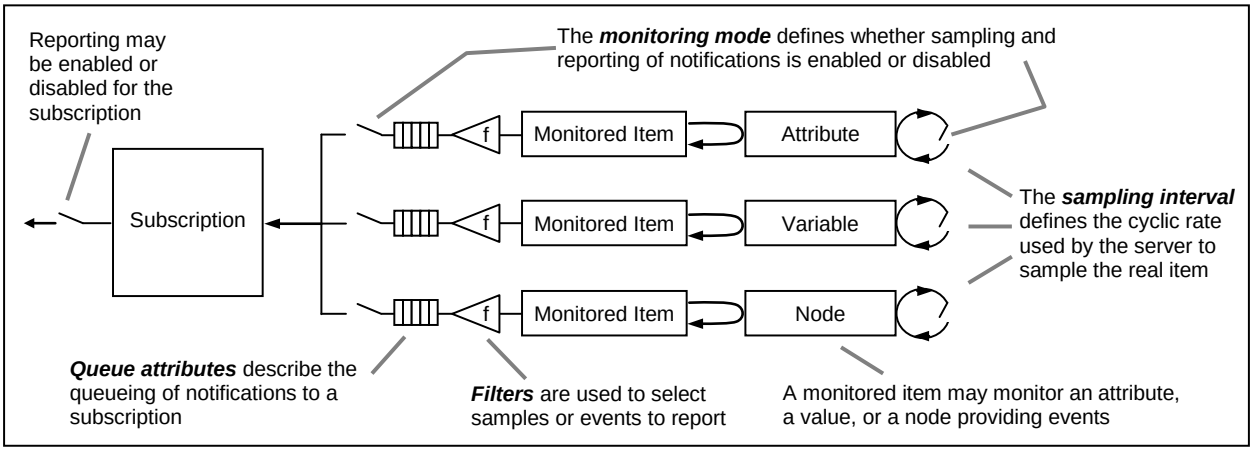


Figure 15 – MonitoredItem Model

Attributes, other than the *Value Attribute*, are only monitored for a change in value. The filter is not used for these *Attributes*. Any change in value for these *Attributes* causes a *Notification* to be generated.

The *Value Attribute* is used when monitoring *Variables*. *Variable* values are monitored for a change in value or a change in their status. The filters defined in this standard (see 7.16.2) and in IEC 62541-8 are used to determine if the value change is large enough to cause a *Notification* to be generated for the *Variable*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Objects and views can be used to monitor *Events*. *Events* are only available from *Nodes* where the *SubscribeToEvents* bit of the *EventNotifier Attribute* is set. The filter defined in this standard (see 7.16.3) is used to determine if an *Event* received from the *Node* is sent to the *Client*. The filter also allows selecting fields of the *EventType* that will be contained in the *Event* such as *EventId*, *EventType*, *SourceNode*, *Time* and *Description*.

IEC 62541-3 describes the *Event* model and the base *EventTypes*.

The *Properties* of the base *EventTypes* and the representation of the base *EventTypes* in the *AddressSpace* are specified in IEC 62541-5.

5.12.1.2 Sampling interval

Each *MonitoredItem* created by the *Client* is assigned a sampling interval that is either inherited from the publishing interval of the *Subscription* or that is defined specifically to override that rate. The sampling interval indicates the fastest rate at which the *Server* should sample its underlying source for data changes.

A *Client* shall define a sampling interval of 0 if it subscibes for *Events*.

The assigned sampling interval defines a “best effort” cyclic rate that the *Server* uses to sample the item from its source. “Best effort” in this context means that the *Server* does its best to sample at this rate. Sampling at rates faster than this rate is acceptable, but not necessary to meet the needs of the *Client*. How the *Server* deals with the sampling rate and how often it actually polls its data source internally is a *Server* implementation detail. However, the time between values returned to the *Client* shall be greater or equal to the sampling interval.

The *Client* may also specify 0 for the sampling interval, which indicates that the *Server* should use the fastest practical rate. It is expected that *Servers* will support only a limited set of sampling intervals to optimize their operation. If the exact interval requested by the *Client* is not supported by the *Server*, then the *Server* assigns to the *MonitoredItem* the most appropriate interval as determined by the *Server*. It returns this assigned interval to the *Client*. The *Server* Capabilities *Object* defined in IEC 62541-5 identifies the sampling intervals supported by the *Server*.

The *Server* may support data that is collected based on a sampling model or generated based on an exception-based model. The fastest supported sampling interval may be equal to 0, which indicates that the data item is exception-based rather than being sampled at some period. Exception-based means that the underlying system does not require sampling and reports changes of the data.

The *Client* may use the revised sampling interval values as a hint for setting the publishing interval as well as the keep alive count of a *Subscription*. If, for example, the smallest revised sampling interval of the *MonitoredItems* is 5 s, then the time before a keep-alive is sent should be longer than 5 s.

Note that, in many cases, the OPC UA *Server* provides access to a decoupled system and therefore has no knowledge of the data update logic. In this case, even though the OPC UA *Server* samples at the negotiated rate, the data might be updated by the underlying system at a much slower rate. In this case, changes can only be detected at this slower rate.

If the behaviour by which the underlying system updates the item is known, it will be available via the *MinimumSamplingInterval Attribute* defined in IEC 62541-3. If the *Server* specifies a value for the *MinimumSamplingInterval Attribute*, it shall always return a *revisedSamplingInterval* that is equal or higher than the *MinimumSamplingInterval* if the *Client* subscribes to the *Value Attribute*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Clients should also be aware that the sampling by the OPC UA *Server* and the update cycle of the underlying system are usually not synchronized. This can cause additional delays in change detection, as illustrated in Figure 16.

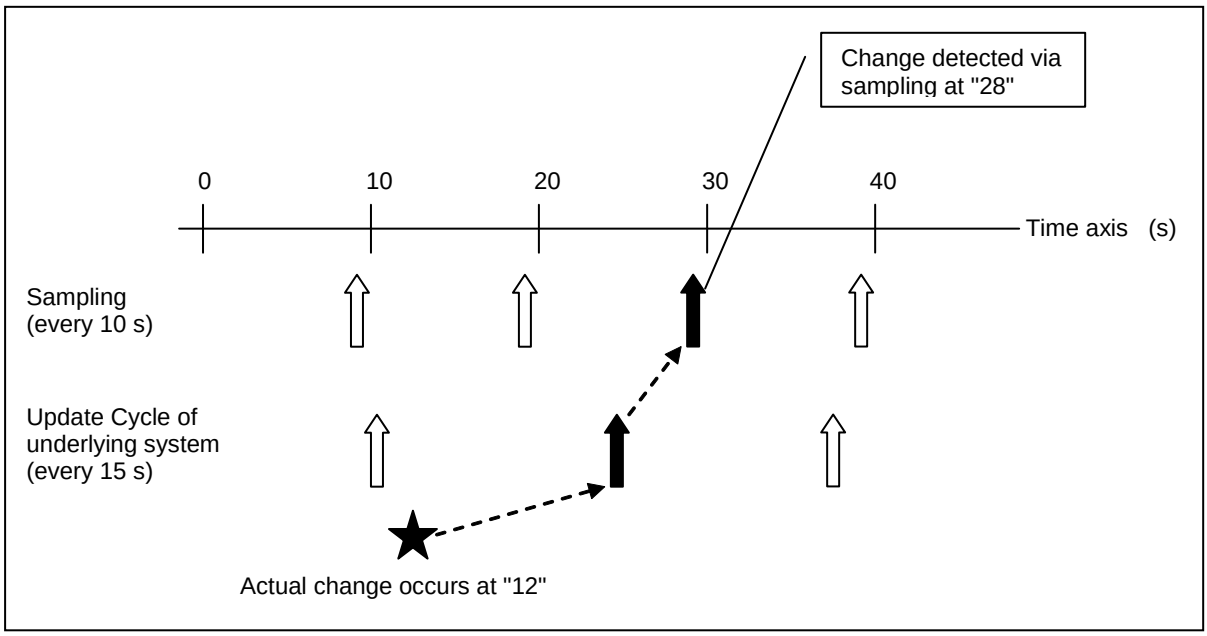


Figure 16 – Typical delay in change detection

5.12.1.3 Monitoring mode

The monitoring mode parameter is used to enable and disable the sampling of a *MonitoredItem*, and also to provide for independently enabling and disabling the reporting of *Notifications*. This capability allows a *MonitoredItem* to be configured to sample, sample and report, or neither. Disabling sampling does not change the values of any of the other *MonitoredItem* parameter, such as its sampling interval.

When a *MonitoredItem* is enabled (i.e. when the *MonitoringMode* is changed from *DISABLED* to *SAMPLING* or *REPORTING*) or it is created in the enabled state, the *Server* shall report the first sample as soon as possible and the time of this sample becomes the starting point for the next sampling interval.

5.12.1.4 Filter

Each time a *MonitoredItem* is sampled, the *Server* evaluates the sample using the filter defined for the *MonitoredItem*. The filter parameter defines the criteria that the *Server* uses to determine if a *Notification* should be generated for the sample. The type of filter is dependent on the type of the item that is being monitored. For example, the *DataChangeFilter* and the *AggregateFilter* are used when monitoring *Variable Values* and the *EventFilter* is used when monitoring *Events*. Sampling and evaluation, including the use of filters, are described in this standard. Additional filters may be defined in other parts of this series of standards.

5.12.1.5 Queue parameters

If the sample passes the filter criteria, a *Notification* is generated and queued for transfer by the *Subscription*. The size of the queue is defined when the *MonitoredItem* is created. When the queue is full and a new *Notification* is received, the *Server* either discards the oldest *Notification* and queues the new one, or it simply discards the new one. The *MonitoredItem* is configured for one of these discard policies when the *MonitoredItem* is created. If a Notification is discarded for a *DataValue*, the *Overflow* bit in the *InfoBits* portion of the *DataValue statusCode* is set.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

If the queue size is one and if the discard policy is to discard the oldest, the queue becomes a buffer that always contains the newest *Notification*. In this case, if the sampling interval of the *MonitoredItem* is faster than the publishing interval of the *Subscription*, the *MonitoredItem* will be over sampling and the *Client* will always receive the most up-to-date value.

On the other hand, the *Client* may want to subscribe to a continuous stream of *Notifications* without any gaps, but does not want them reported at the sampling interval. In this case, the *MonitoredItem* would be created with a queue size large enough to hold all *Notifications* generated between two consecutive publishing cycles. Then, at each publishing cycle, the *Subscription* would send all *Notifications* queued for the *MonitoredItem* to the *Client*. The *Server* shall return *Notifications* for any particular item in the same order they are in the queue.

The *Server* may be sampling at a faster rate than the sampling interval to support other *Clients*; the *Client* should only expect values at the negotiated sampling interval. The *Server* may deliver fewer values than dictated by the sampling interval, based on the filter and implementation constraints. If a *DataChangeFilter* is configured for a *MonitoredItem*, it is always applied to the newest value in the queue compared to the current sample.

If, for example, the *AbsoluteDeadband* in the *DataChangeFilter* is “10”, the queue could consist of values in the following order:

- 100
- 111
- 101
- 89
- 100

Queuing of data may result in unexpected behaviour when using a *Deadband* filter and the number of encountered changes is larger than the number of values that can be maintained. It is realistically possible that, due to the discard policy “`discardOldest=TRUE`”, the new first value in the queue may not exceed the *Deadband* limit of the previous value sent to the *Client*.

The queue size is the maximum value supported by the *Server* when monitoring *Events*. In this case, the *Server* is responsible for the *Event* buffer. If *Events* are lost, an *Event* of the type *EventQueueOverflowEventQueueOverflowEventType* is generated and placed. This Event is generated when the first Event has to be discarded on a MonitoredItem subscribing for Events. It is put into the Queue of the MonitoredItem in addition to the size of the Queue defined for this MonitoredItem without discarding any other Event. If discardOldest is set to TRUE it is put at the beginning of the queue and is never discarded, otherwise at the end. An aggregating *Server* shall not pass on such an *Event*. It shall be handled like other connection error scenarios.

5.12.1.6 Triggering model

The *MonitoredItems Service* allows adding items that are reported only when some other item (the triggering item) triggers. This is done by creating links between the triggered items and the items to report. The monitoring mode of the items to report is set to sampling-only so that it will sample and queue *Notifications* without reporting them. Figure 17 illustrates this concept.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

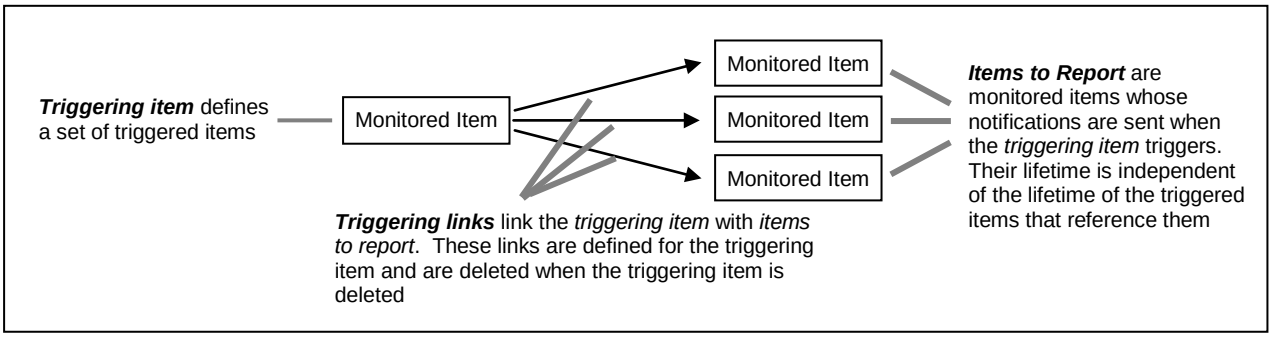


Figure 17 – Triggering Model

The triggering mechanism is a useful feature that allows *Clients* to reduce the data volume on the wire by configuring some items to sample frequently but only report when some other *Event* happens.

The following triggering behaviours are specified.

a) The monitoring mode of the triggering item indicates that reporting is disabled. In this case, the triggering item is not reported when the triggering item triggers.

b) The monitoring mode of the triggering item indicates that reporting is enabled. In this case, the triggering item is reported when the triggering item triggers.

c) The monitoring mode of the item to report indicates that reporting is disabled. In this case, the item to report is reported when the triggering item triggers.

d) The monitoring mode of the item to report indicates that reporting is enabled. In this case, the item to report is reported only once (when the item to report triggers), effectively causing the triggering item to be ignored.

Clients create and delete triggering links between a triggering item and a set of items to report. If the *MonitoredItem* that represents an item to report is deleted before its associated triggering link is deleted, the triggering link is also deleted, but the triggering item is otherwise unaffected.

Deletion of a *MonitoredItem* should not be confused with the removal of the *Attribute* that it monitors. If the *Node* that contains the *Attribute* being monitored is deleted, the *MonitoredItem* generates a *Notification* with a *StatusCode Bad_UnknownNodeId* that indicates the deletion, but the *MonitoredItem* is not deleted.

5.12.2 CreateMonitoredItems

5.12.2.1 Description

This *Service* is used to create and add one or more *MonitoredItems* to a *Subscription*. A *MonitoredItem* is deleted automatically by the *Server* when the *Subscription* is deleted. Deleting a *MonitoredItem* causes its entire set of triggered item links to be deleted, but has no effect on the *MonitoredItems* referenced by the triggered items.

Calling the *CreateMonitoredItems Service* repetitively to add a small number of *MonitoredItems* each time may adversely affect the performance of the *Server*. Instead, *Clients* should add a complete set of *MonitoredItems* to a *Subscription* whenever possible.

When a *MonitoredItem* is added, the *Server* performs initialization processing for it. The initialization processing is defined by the *Notification* type of the item being monitored. *Notification* types are specified in this standard and in the Access Type Specification parts of this series of standards, such as IEC 62541-8. See Clause 4 of IEC/TR 62541-1 for a description of the Access Type Parts.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

When a user adds a monitored item that the user is denied read access to, the add operation for the item shall succeed and the bad status Bad_NotReadable or Bad_UserAccessDenied shall be returned in the Publish response. This is the same behaviour for the case where the access rights are changed after the call to *CreateMonitoredItem*. If the access rights change to read rights, the *Server* shall start sending data for the *MonitoredItem*.

The return diagnostic info setting in the request header of the *CreateMonitoredItems* or the last *ModifyMonitoredItems Service* is applied to the *Monitored Items* and is used as the diagnostic information settings when sending Notifications in the *Publish* response.

5.12.2.2 Parameters

Table 64 defines the parameters for the *Service*.

Table 64 – CreateMonitoredItems Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* that will report *Notifications* for this *MonitoredItem* (see 7.13 for *IntegerId* definition).
timestampsToReturn	enum TimestampsToReturn	An enumeration that specifies the timestamp *Attributes* to be transmitted for each *MonitoredItem*. The *TimestampsToReturn* enumeration is defined in 7.34. When monitoring *Events*, this applies only to *Event* fields that are of type *DataValue*.
itemsToCreate []	MonitoredItemCreateRequest	A list of *MonitoredItems* to be created and assigned to the specified *Subscription*.
itemToMonitor	ReadValueId	Identifies an item in the *AddressSpace* to monitor. To monitor for *Events*, the *attributeId* element of the *ReadValueId* structure is the id of the *EventNotifier Attribute*. The *ReadValueId* type is defined in 7.23.
monitoringMode	enum MonitoringMode	The monitoring mode to be set for the *MonitoredItem*. The *MonitoringMode* enumeration is defined in 7.17.
requestedParameters	MonitoringParameters	The requested monitoring parameters. *Servers* negotiate the values of these parameters based on the *Subscription* and the capabilities of the *Server*. The *MonitoringParameters* type is defined in 7.15.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	MonitoredItemCreateResult	List of results for the *MonitoredItems* to create. The size and order of the list matches the size and order of the *itemsToCreate* request parameter.
statusCode	StatusCode	*StatusCode* for the *MonitoredItem* to create (see 7.33 for *StatusCode* definition).
monitoredItemId	IntegerId	*Server*-assigned id for the *MonitoredItem* (see 7.13 for *IntegerId* definition). This id is unique within the *Subscription*, but might not be unique within the *Server* or *Session*. This parameter is present only if the *statusCode* indicates that the *MonitoredItem* was successfully created.
revisedSamplingInterval	Duration	The actual sampling interval that the *Server* will use. This value is based on a number of factors, including capabilities of the underlying system. The Server shall always return a *revisedSamplingInterval* that is equal or higher than the *requestedSamplingInterval*. If the *requestedSamplingInterval* is higher than the maximum sampling interval supported by the *Server*, the maximum sampling interval is returned.
revisedQueueSize	Counter	The actual queue size that the *Server* will use.
filterResult	Extensible Parameter MonitoringFilterResult	Contains any revised parameter values or error results associated with the *MonitoringFilter* specified in the request. This parameter may be omitted if no errors occurred. The *MonitoringFilterResult* parameter type is an extensible parameter type specified in 7.16.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *MonitoredItems* to create (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *itemsToCreate* 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.12.2.3 Service results

Table 65 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 65 – CreateMonitoredItems 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.
Bad_TimestampsToReturnInvalid	See Table 165 for the description of this result code.
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.

5.12.2.4 StatusCodes

Table 66 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 66 – CreateMonitoredItems Operation Level Result Codes

Symbolic Id	Description
Bad_MonitoringModeInvalid	See Table 166 for the description of this result code.
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_AttributeIdInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.
Bad_IndexRangeNoData	See Table 166 for the description of this result code.
Bad_DataEncodingInvalid	See Table 166 for the description of this result code.
Bad_DataEncodingUnsupported	See Table 166 for the description of this result code.
Bad_UserAccessDenied	See Table 165 for the description of this result code.
Bad_MonitoredItemFilterInvalid	See Table 166 for the description of this result code.
Bad_MonitoredItemFilterUnsupported	See Table 166 for the description of this result code.
Bad_FilterNotAllowed	See Table 165 for the description of this result code.

5.12.3 ModifyMonitoredItems

5.12.3.1 Description

This *Service* is used to modify *MonitoredItems* of a *Subscription*. Changes to the sampling interval and filter take effect at the beginning of the next sampling interval (the next time the sampling timer expires).

The return diagnostic info setting in the request header of the *CreateMonitoredItems* or the last *ModifyMonitoredItems Service* is applied to the *Monitored Items* and is used as the diagnostic information settings when sending Notifications in the *Publish* response.

5.12.3.2 Parameters

Table 67 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 67 – ModifyMonitoredItems Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* used to qualify the *monitoredItemId* (see 7.13 for *IntegerId* definition).
timestampsToReturn	enum TimestampsToReturn	An enumeration that specifies the timestamp *Attributes* to be transmitted for each *MonitoredItem* to be modified. The *TimestampsToReturn* enumeration is defined in 7.34. When monitoring *Events*, this applies only to *Event* fields that are of type *DataValue*.
itemsToModify []	MonitoredItemModifyRequest	The list of *MonitoredItems* to modify.
monitoredItemId	IntegerId	*Server*-assigned id for the *MonitoredItem*.
requestedParameters	Monitoring Parameters	The requested values for the monitoring parameters. The *MonitoringParameters* type is defined in 7.14.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	MonitoredItemModifyResult	List of results for the *MonitoredItems* to modify. The size and order of the list matches the size and order of the *itemsToModify* request parameter.
statusCode	StatusCode	*StatusCode* for the *MonitoredItem* to be modified (see 7.33 for *StatusCode* definition).
revisedSampling Interval	Duration	The actual sampling interval that the *Server* will use. The *Server* returns the value it will actually use for the sampling interval. This value is based on a number of factors, including capabilities of the underlying system. The Server shall always return a *revisedSamplingInterval* that is equal or higher than the *requestedSamplingInterval*. If the *requestedSamplingInterval* is higher than the maximum sampling interval supported by the *Server*, the maximum sampling interval is returned.
revisedQueueSize	Counter	The actual queue size that the *Server* will use.
filterResult	Extensible Parameter MonitoringFilterResult	Contains any revised parameter values or error results associated with the *MonitoringFilter* specified in the request. This parameter may be omitted if no errors occurred. The *MonitoringFilterResult* parameter type is an extensible parameter type specified in 7.16.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *MonitoredItems* to modify (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *itemsToModify* 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.

5.12.3.3 Service results

Table 68 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 68 – ModifyMonitoredItems 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.
Bad_TimestampsToReturnInvalid	See Table 165 for the description of this result code.
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.

5.12.3.4 StatusCodes

Table 69 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 69 – ModifyMonitoredItems Operation Level Result Codes

Symbolic Id	Description
Bad_MonitoredItemIdInvalid	See Table 166 for the description of this result code.
Bad_MonitoredItemFilterInvalid	See Table 166 for the description of this result code.
Bad_MonitoredItemFilterUnsupported	See Table 166 for the description of this result code.
Bad_FilterNotAllowed	See Table 165 for the description of this result code.

5.12.4 SetMonitoringMode

5.12.4.1 Description

This *Service* is used to set the monitoring mode for one or more *MonitoredItems* of a *Subscription*. Setting the mode to DISABLED causes all queued *Notifications* to be deleted.

5.12.4.2 Parameters

Table 70 defines the parameters for the *Service*.

Table 70 – SetMonitoringMode Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* used to qualify the *monitoredItemIds* (see 7.13 for *IntegerId* definition).
monitoringMode	enum MonitoringMode	The monitoring mode to be set for the *MonitoredItems*. The *MonitoringMode* enumeration is defined in 7.17.
monitoredItemIds []	IntegerId	List of *Server*-assigned ids for the *MonitoredItems*.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *MonitoredItems* to enable/disable (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *monitoredItemIds* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *MonitoredItems* to enable/disable (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *monitoredItemIds* 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.

5.12.4.3 Service results

Table 71 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 71 – SetMonitoringMode 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.
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.
Bad_MonitoringModeInvalid	See Table 166 for the description of this result code.

5.12.4.4 StatusCodes

Table 72 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Table 72 – SetMonitoringMode Operation Level Result Codes

Value	Description
Bad_MonitoredItemIdInvalid	See Table 166 for the description of this result code.

5.12.5 SetTriggering

5.12.5.1 Description

This *Service* is used to create and delete triggering links for a triggering item. The triggering item and the items to report shall belong to the same *Subscription*.

Each triggering link links a triggering item to an item to report. Each link is represented by the *MonitoredItem* id for the item to report. An error code is returned if this id is invalid.

5.12.5.2 Parameters

Table 73 defines the parameters for the *Service*.

Table 73 – SetTriggering Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* that contains the triggering item and the items to report (see 7.13 for *IntegerId* definition).
triggeringItemId	IntegerId	*Server*-assigned id for the *MonitoredItem* used as the triggering item.
linksToAdd []	IntegerId	The list of *Server*-assigned ids of the items to report that are to be added as triggering links.
linksToRemove []	IntegerId	The list of *Server*-assigned ids of the items to report for the triggering links to be deleted.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
addResults []	StatusCode	List of *StatusCodes* for the items to add (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *linksToAdd* parameter specified in the request.
addDiagnosticInfos []	Diagnostic Info	List of diagnostic information for the links to add (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *linksToAdd* 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.
removeResults []	StatusCode	List of *StatusCodes* for the items to delete. The size and order of the list matches the size and order of the *linksToDelete* parameter specified in the request.
removeDiagnosticInfos []	Diagnostic Info	List of diagnostic information for the links to delete. The size and order of the list matches the size and order of the *linksToDelete* 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.

5.12.5.3 Service results

Table 74 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in 7.33.

Table 74 – SetTriggering 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.
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.
Bad_MonitoredItemIdInvalid	See Table 166 for the description of this result code.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.12.5.4 StatusCodes

Table 75 defines values for the results parameters that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 75 – SetTriggering Operation Level Result Codes

Symbolic Id	Description
Bad_MonitoredItemIdInvalid	See Table 166 for the description of this result code.

5.12.6 DeleteMonitoredItems

5.12.6.1 Description

This *Service* is used to remove one or more *MonitoredItems* of a *Subscription*. When a *MonitoredItem* is deleted, its triggered item links are also deleted.

Successful removal of a *MonitoredItem*, however, might not remove *Notifications* for the *MonitoredItem* that are in the process of being sent by the *Subscription*. Therefore, *Clients* may receive *Notifications* for the *MonitoredItem* after they have received a positive response that the *MonitoredItem* has been deleted.

5.12.6.2 Parameters

Table 76 defines the parameters for the *Service*.

Table 76 – DeleteMonitoredItems Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* that contains the *MonitoredItems* to be deleted (see 7.13 for *IntegerId* definition).
monitoredItemIds []	IntegerId	List of *Server*-assigned ids for the *MonitoredItems* to be deleted.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *MonitoredItems* to delete (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *monitoredItemIds* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *MonitoredItems* to delete (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *monitoredItemIds* 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.

5.12.6.3 Service results

Table 77 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 77 – DeleteMonitoredItems 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.
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.

5.12.6.4 StatusCodes

Table 78 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 78 – DeleteMonitoredItems Operation Level Result Codes

Symbolic Id	Description
Bad_MonitoredItemIdInvalid	See Table 166 for the description of this result code.

5.13 Subscription Service Set

5.13.1 Subscription model

5.13.1.1 Description

Subscriptions are used to report *Notifications* to the *Client*. Their general behaviour is summarized below. Their precise behaviour is described in 5.13.1.2.

a) *Subscriptions* have a set of *MonitoredItems* assigned to them by the *Client*. *MonitoredItems* generate *Notifications* that are to be reported to the *Client* by the *Subscription* (see 5.12.1 for a description of *MonitoredItems*).

b) *Subscriptions* have a publishing interval. The publishing interval of a *Subscription* defines the cyclic rate at which the *Subscription* executes. Each time it executes, it attempts to send a *NotificationMessage* to the *Client*. *NotificationMessages* contain *Notifications* that have not yet been reported to the *Client*.

c) *NotificationMessages* are sent to the *Client* in response to *Publish* requests. *Publish* requests are normally queued to the *Session* as they are received, and one is dequeued and processed by a subscription related to this *Session* for each publishing cycle, if there are *Notifications* to report. When there are not, the *Publish* request is not dequeued from the *Session*, and the *Server* waits until the next cycle and checks again for *Notifications*.

d) At the beginning of a cycle, if there are *Notifications* to send but there are no *Publish* requests queued, the *Server* enters a wait state for a *Publish* request to be received. When one is received, it is processed immediately without waiting for the next publishing cycle.

e) *NotificationMessages* are uniquely identified by sequence numbers that enable *Clients* to detect missed *Messages*. The publishing interval also defines the default sampling interval for its *MonitoredItems*.

f) *Subscriptions* have a keep-alive counter that counts the number of consecutive publishing cycles in which there have been no *Notifications* to report to the *Client*. When the maximum keep-alive count is reached, a *Publish* request is dequeued and used to return a keep-alive *Message*. This keep-alive *Message* informs the *Client* that the *Subscription* is still active. Each keep-alive *Message* is a response to a Publish request in which the *notificationMessage* parameter does not contain any Notifications and that contains the sequence number of the next *NotificationMessage* that is to be sent. In the subclauses that follow, the term *NotificationMessage* refers to a response to a Publish request in which the *notificationMessage* parameter actually contains one or more *Notifications*, as opposed to a *keep-alive Message* in which this parameter contains no *Notifications*. The maximum keep-alive count is set by the *Client* during *Subscription* creation and may be subsequently modified using the *ModifySubscription Service*. Similar to *Notification* processing described in c) above, if there are no *Publish* requests queued, the *Server* waits for the next one to be received and sends the keep-alive immediately without waiting for the next publishing cycle.

g) Publishing by a *Subscription* may be enabled or disabled by the *Client* when created, or subsequently using the *SetPublishingMode Service*. Disabling causes the *Subscription* to cease sending *NotificationMessages* to the *Client*. However, the *Subscription* continues to execute cyclically and continues to send keep-alive *Messages* to the *Client*.

h) *Subscriptions* have a lifetime counter that counts the number of consecutive publishing cycles in which there have been no *Publish* requests received from the *Client*. When this counter reaches the value calculated for the lifetime of a *Subscription* based on the MaxKeepAliveCount parameter in the *CreateSubscription Service* (5.13.2), the *Subscription* is closed. Closing the *Subscription* causes its *MonitoredItems* to be deleted. In addition, the *Server* shall issue a *StatusChangeNotification notificationMessage* with the

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

status code Bad_Timeout. The *StatusChangeNotification notificationMessage* type is defined in 7.19.4.

i) *Subscriptions* maintain a retransmission queue of sent *NotificationMessages*. *NotificationMessages* are retained in this queue until they are acknowledged or until they have been in the queue for a minimum of one keep-alive interval. *Clients* are required to acknowledge *NotificationMessages* as they are received.

The sequence number is an unsigned 32-bit integer that is incremented by one for each *NotificationMessage* sent. The value 0 is never used for the sequence number. The first *NotificationMessage* sent on a *Subscription* has a sequence number of 1. If the sequence number rolls over, it rolls over to 1.

When a *Subscription* is created, the first *Message* is sent at the end of the first publishing cycle to inform the *Client* that the *Subscription* is operational. A *NotificationMessage* is sent if there are *Notifications* ready to be reported. If there are none, a keep-alive *Message* is sent instead that contains a sequence number of 1, indicating that the first *NotificationMessage* has not yet been sent. This is the only time a keep-alive *Message* is sent without waiting for the maximum keep-alive count to be reached, as specified in f) above.

The value of the sequence number is never reset during the lifetime of a *Subscription*. Therefore, the same sequence number shall not be reused on a *Subscription* until over four billion *NotificationMessages* have been sent. At a continuous rate of one thousand *NotificationMessages* per second on a given *Subscription*, it would take roughly fifty days for the same sequence number to be reused. This allows *Clients* to safely treat sequence numbers as unique.

Sequence numbers are also used by *Clients* to acknowledge the receipt of *NotificationMessages*. *Publish* requests allow the *Client* to acknowledge all *Notifications* up to a specific sequence number and to acknowledge the sequence number of the last *NotificationMessage* received. One or more gaps may exist in between. Acknowledgements allow the *Server* to delete *NotificationMessages* from its retransmission queue.

Clients may ask for retransmission of selected *NotificationMessages* using the Republish *Service*. This *Service* returns the requested *Message*.

5.13.1.2 State table

The state table formally describes the operation of the *Subscription*. The following model of operations is described by this state table. This description applies when publishing is enabled or disabled for the *Subscription*.

After creation of the *Subscription*, the *Server* starts the publishing timer and restarts it whenever it expires. If the timer expires the number of times defined for the *Subscription* lifetime without having received a *Subscription Service* request from the *Client*, the *Subscription* assumes that the *Client* is no longer present, and terminates.

Clients send *Publish* requests to *Servers* to receive *Notifications*. *Publish* requests are not directed to any one *Subscription* and, therefore, may be used by any *Subscription*. Each contains acknowledgements for one or more *Subscriptions*. These acknowledgements are processed when the *Publish* request is received. The *Server* then queues the request in a queue shared by all *Subscriptions*, except in the following cases.

a) The previous *Publish* response indicated that there were still more *Notifications* ready to be transferred and there were no more *Publish* requests queued to transfer them.

b) The publishing timer of a *Subscription* expired and there were either *Notifications* to be sent or a keep-alive *Message* to be sent.

In these cases, the newly received *Publish* request is processed immediately by the first *Subscription* to encounter either case a) or case b).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Each time the publishing timer expires, it is immediately reset. If there are *Notifications* or a keep-alive *Message* to be sent, it dequeues and processes a *Publish* request. When a *Subscription* processes a *Publish* request, it accesses the queues of its *MonitoredItems* and dequeues its *Notifications*, if any. It returns these *Notifications* in the response, setting the *moreNotifications* flag if it was not able to return all available *Notifications* in the response.

If there were *Notifications* or a keep-alive *Message* to be sent but there were no *Publish* requests queued, the *Subscription* assumes that the *Publish* request is late and waits for the next *Publish* request to be received, as described in case b).

If the *Subscription* is disabled when the publishing timer expires or if there are no *Notifications* available, it enters the keep-alive state and sets the keep-alive counter to its maximum value as defined for the *Subscription*.

While in keep-alive state, it checks for *Notifications* each time the publishing timer expires. If one or more notifications have been generated, a *Publish* request is dequeued and a *NotificationMessage* is returned in the response. However, if the publishing timer expires without a *Notification* becoming available, a *Publish* request is dequeued and a keep-alive *Message* is returned in the response. The *Subscription* then returns to the normal state of waiting for the publishing timer to expire again. If, in either of these cases, there are no *Publish* requests queued, the *Subscription* waits for the next *Publish* request to be received, as described in case b).

The *Subscription* states are defined in Table 79.

Table 79 – Subscription States

State	Description
CLOSED	The *Subscription* has not yet been created or has terminated.
CREATING	The *Subscription* is being created.
NORMAL	The *Subscription* is cyclically checking for *Notifications* from its *MonitoredItems*. The keep-alive counter is not used in this state.
LATE	The publishing timer has expired and there are *Notifications* available or a keep-alive *Message* is ready to be sent, but there are no *Publish* requests queued. When in this state, the next *Publish* request is processed when it is received. The keep-alive counter is not used in this state.
KEEPALIVE	The *Subscription* is cyclically checking for *Notifications* from its *MonitoredItems* or for the keep-alive counter to count down to 0 from its maximum.

The state table is described in Table 80. The following rules and conventions apply.

a) *Events* represent the receipt of *Service* requests and the occurrence internal *Events*, such as timer expirations.

b) *Service* requests *Events* may be accompanied by conditions that test *Service* parameter values. Parameter names begin with a lower case letter.

c) Internal *Events* may be accompanied by conditions that test state *Variable* values. State *Variables* are defined in 5.13.1.3. They begin with an upper case letter.

d) *Service* request and internal *Events* may be accompanied by conditions represented by functions whose return value is tested. Functions are identified by “()” after their name. They are described in 5.13.1.4.

e) When an *Event* is received, the first transition for the current state is located and the transitions are searched sequentially for the first transition that meets the *Event* or conditions criteria. If none are found, the *Event* is ignored.

f) Actions are described by functions and state *Variable* manipulations.

g) The LifetimeTimerExpires *Event* is triggered when its corresponding counter reaches zero.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Table 80 – Subscription State Table

#	Current State	Event/Conditions	Action	Next State
1	CLOSED	Receive CreateSubscription Request	CreateSubscription()	CREATING
2	CREATING	CreateSubscription fails	ReturnNegativeResponse()	CLOSED
3	CREATING	CreateSubscription succeeds	InitializeSubscription() MessageSent = FALSE ReturnResponse()	NORMAL
4	NORMAL	Receive *Publish* Request && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && MoreNotifications == FALSE))	ResetLifetimeCounter() DeleteAckedNotificationMsgs() EnqueuePublishingReq()	NORMAL
5	NORMAL	Receive *Publish* Request && PublishingEnabled == TRUE && MoreNotifications == TRUE	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnNotifications() MessageSent = TRUE	NORMAL
6	NORMAL	PublishingTimer Expires && PublishingReqQueued == TRUE && PublishingEnabled == TRUE && NotificationsAvailable == TRUE	StartPublishingTimer() DequeuePublishReq() ReturnNotifications() MessageSent == TRUE	NORMAL
7	NORMAL	PublishingTimer Expires && PublishingReqQueued == TRUE && MessageSent == FALSE && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() DequeuePublishReq() ReturnKeepAlive() MessageSent == TRUE	NORMAL
8	NORMAL	PublishingTimer Expires && PublishingReqQueued == FALSE && (MessageSent == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == TRUE))	StartPublishingTimer()	LATE
9	NORMAL	PublishingTimer Expires && MessageSent == TRUE && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() ResetKeepAliveCounter()	KEEPALIVE
10	LATE	Receive *Publish* Request && PublishingEnabled == TRUE && (NotificationsAvailable == TRUE || MoreNotifications == TRUE)	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnNotifications() MessageSent = TRUE	NORMAL
11	LATE	Receive *Publish* Request && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE && MoreNotifications == FALSE))	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnKeepAlive() MessageSent = TRUE	KEEPALIVE
12	LATE	PublishingTimer Expires	StartPublishingTimer()	LATE

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

#	Current State	Event/Conditions	Action	Next State
13	KEEPALIVE	Receive *Publish* Request	ResetLifetimeCounter() DeleteAckedNotificationMsgs() EnqueuePublishingReq()	KEEPALIVE
14	KEEPALIVE	PublishingTimer Expires && PublishingEnabled == TRUE && NotificationsAvailable == TRUE && PublishingReqQueued == TRUE	StartPublishingTimer() DequeuePublishReq() ReturnNotifications() MessageSent == TRUE	NORMAL
15	KEEPALIVE	PublishingTimer Expires && PublishingReqQueued == TRUE && KeepAliveCounter == 1 && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE)	StartPublishingTimer() DequeuePublishReq() ReturnKeepAlive() ResetKeepAliveCounter()	KEEPALIVE
16	KEEPALIVE	PublishingTimer Expires && KeepAliveCounter > 1 && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() KeepAliveCounter--	KEEPALIVE
17	KEEPALIVE	PublishingTimer Expires && PublishingReqQueued == FALSE && (KeepAliveCounter == 1 || (KeepAliveCounter > 1 && PublishingEnabled == TRUE && NotificationsAvailable == TRUE))	StartPublishingTimer()	LATE
18	NORMAL || LATE || KEEPALIVE	Receive ModifySubscription Request	ResetLifetimeCounter() UpdateSubscriptionParams() ReturnResponse()	SAME
19	NORMAL || LATE || KEEPALIVE	Receive SetPublishingMode Request	ResetLifetimeCounter() SetPublishingEnabled() MoreNotifications = FALSE ReturnResponse()	SAME
20	NORMAL || LATE || KEEPALIVE	Receive Republish Request && RequestedMessageFound == TRUE	ResetLifetimeCounter() ReturnResponse()	SAME
21	NORMAL || LATE || KEEPALIVE	Receive Republish Request && RequestedMessageFound == FALSE	ResetLifetimeCounter() ReturnNegativeResponse()	SAME
22	NORMAL || LATE || KEEPALIVE	Receive TransferSubscriptions Request && SessionChanged() == FALSE	ResetLifetimeCounter() ReturnNegativeResponse ()	SAME
23	NORMAL || LATE || KEEPALIVE	Receive TransferSubscriptions Request && SessionChanged() == TRUE && ClientValidated() ==TRUE	SetSession() ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnResponse() IssueStatusChangeNotification()	SAME
24	NORMAL || LATE || KEEPALIVE	Receive TransferSubscriptions Request && SessionChanged() == TRUE && ClientValidated() == FALSE	ReturnNegativeResponse()	SAME
25	NORMAL || LATE || KEEPALIVE	Receive DeleteSubscriptions Request && SubscriptionAssignedToClient ==TRUE	DeleteMonitoredItems() DeleteClientPublReqQueue()	CLOSED

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

#	Current State	Event/Conditions	Action	Next State
26	NORMAL || LATE || KEEPALIVE	Receive DeleteSubscriptions Request && SubscriptionAssignedToClient ==FALSE	ResetLifetimeCounter() ReturnNegativeResponse()	SAME
27	NORMAL || LATE || KEEPALIVE	LifetimeCounter == 1 The LifetimeCounter is decremented if PublishingTimer expires and PublishingReqQueued == FALSE The LifetimeCounter is reset if PublishingReqQueued == TRUE.	DeleteMonitoredItems() IssueStatusChangeNotification()	CLOSED

5.13.1.3 State Variables and parameters

The state *Variables* are defined alphabetically in Table 81.

Table 81 – State variables and parameters

State Variable	Description
MoreNotifications	A boolean value that is set to TRUE only by the CreateNotificationMsg() when there were too many *Notifications* for a single *NotificationMessage*.
LatePublishRequest	A boolean value that is set to TRUE to reflect that, the last time the publishing timer expired, there were no *Publish* requests queued.
LifetimeCounter	A value that contains the number of consecutive publishing timer expirations without *Client* activity before the *Subscription* is terminated.
MessageSent	A boolean value that is set to TRUE to mean that either a *NotificationMessage* or a keep-alive *Message* has been sent on the *Subscription*. It is a flag that is used to ensure that either a *NotificationMessage* or a keep-alive *Message* is sent out the first time the publishing timer expires.
NotificationsAvailable	A boolean value that is set to TRUE only when there is at least one *MonitoredItem* that is in the reporting mode and that has a *Notification* queued or there is at least one item to report whose triggering item has triggered and that has a *Notification* queued. The transition of this state *Variable* from FALSE to TRUE creates the “New *Notification* Queued” *Event* in the state table.
PublishingEnabled	The parameter that requests publishing to be enabled or disabled.
PublishingReqQueued	A boolean value that is set to TRUE only when there is a *Publish* request *Message* enqueued to the *Subscription*.
RequestedMessageFound	A boolean value that is set to TRUE only when the *Message* requested to be retransmitted was found in the retransmission queue.
SeqNum	The value that records the value of the sequence number used in *NotificationMessages.*
SubscriptionAssignedToClient	A boolean value that is set to TRUE only when the *Subscription* requested to be deleted is assigned to the *Client* that issued the request. A *Subscription* is assigned to the *Client* that created it. That assignment can only be changed through successful completion of the TransferSubscriptions *Service*.

5.13.1.4 Functions

The action functions are defined alphabetically in Table 82.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 82 – Functions

State	Description
BindSession()	Bind the *Client Session* associated with the *Subscription* to the *Client Session* used to send the *Service* being processed. If this was the last *Subscription* bound to the previous *Client*, clear the *Publish* request queue of all *Publish* requests sent by the previous *Client* and return negative responses for each.
ClientValidated()	A boolean function that returns TRUE only when the *Client* that is submitting a TransferSubscriptions request is operating on behalf of the same user and supports the same *Profiles* as the *Client* of the previous *Session*.
CreateNotificationMsg()	Increment the SeqNum and create a *NotificationMessage* from the *MonitoredItems* assigned to the *Subscription*. Save the newly-created *NotificationMessage* in the retransmission queue. If all available *Notifications* can be sent in the *Publish* response, the MoreNotifications state *Variable* is set to FALSE. Otherwise, it is set to TRUE.
CreateSubscription()	Attempt to create the *Subscription*.
DeleteAckedNotificationMsgs()	Delete the *NotificationMessages* from the retransmission queue that were acknowledged by the request.
DeleteClientPublReqQueue()	Clear the *Publish* request queue for the *Client* that is sending the DeleteSubscriptions request, if there are no more *Subscriptions* assigned to that *Client*.
DeleteMonitoredItems()	Delete all *MonitoredItems* assigned to the *Subscription.*
DequeuePublishReq()	Dequeue a publishing request in first-in first-out order. Validate if the publish request is still valid by checking the timeoutHint in the RequestHeader. If the request timed out, send a *Bad_Timeout* service result for the request and dequeue another publish request.
EnqueuePublishingReq()	Enqueue the publishing request.
InitializeSubscription()	ResetLifetimeCounter() MoreNotifications = FALSE PublishRateChange = FALSE PublishingEnabled = value of publishingEnabled parameter in the CreateSubscription request PublishingReqQueued = FALSE SeqNum = 0 SetSession() StartPublishingTimer()
IssueStatusChangeNotification()	Issue a *StatusChangeNotification notificationMessage* with a status code for the status change of the Subscription. The *StatusChangeNotification notificationMessage* type is defined in 7.19.4. Bad_Timeout status code is used if the lifetime expires and Good_SubscriptionTransferred is used if the Subscriptions was transferred to another Session.
ResetKeepAliveCounter()	Reset the keep-alive counter to the maximum keep-alive count of the *Subscription*. The maximum keep-alive count is set by the *Client* when the *Subscription* is created and may be modified using the ModifySubscription *Service*.
ResetLifetimeCounter()	Reset the LifetimeCounter *Variable* to the value specified for the lifetime of a *Subscription* in the CreateSubscription *Service* (5.13.2).
ReturnKeepAlive()	CreateKeepAliveMsg() ReturnResponse()
ReturnNegativeResponse ()	Return a *Service* response indicating the appropriate *Service* level error. No parameters are returned other than the responseHeader that contains the *Service* level *StatusCode*.

ReturnNotifications()	CreateNotificationMsg()
	ReturnResponse()
	If (MoreNotifications == TRUE) && (PublishingReqQueued == TRUE)
	{
	DequeuePublishReq()
	Loop through this function again
	}

ReturnResponse()	Return the appropriate response, setting the appropriate parameter values and *StatusCodes* defined for the *Service*.
SessionChanged()	A boolean function that returns TRUE only when the *Session* used to send a TransferSubscriptions request is different from the *Client Session* currently associated with the *Subscription*.
SetPublishingEnabled ()	Set the PublishingEnabled state *Variable* to the value of the publishingEnabled parameter received in the request.
SetSession	Set the *Session* information for the *Subscription* to match the *Session* on which the TransferSubscriptions request was issued.
StartPublishingTimer()	Start or restart the publishing timer and decrement the LifetimeCounter *Variable*.
UpdateSubscriptionParams()	Negotiate and update the *Subscription* parameters. If the new keep-alive interval is less than the current value of the keep-alive counter, perform ResetKeepAliveCounter() and ResetLifetimeCounter().

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.2 CreateSubscription

5.13.2.1 Description

This *Service* is used to create a *Subscription*. *Subscriptions* monitor a set of *MonitoredItems* for *Notifications* and return them to the *Client* in response to *Publish* requests.

5.13.2.2 Parameters

Table 83 defines the parameters for the *Service*.

Table 83 – CreateSubscription Service Parameters

Name	Type	Description
Request		
requestHeader	Request Header	Common request parameters (see 7.26 for *RequestHeader* definition).
requestedPublishing Interval	Duration	This interval defines the cyclic rate that the *Subscription* is being requested to return *Notifications* to the *Client*. This interval is expressed in milliseconds. This interval is represented by the publishing timer in the *Subscription* state table (see 5.13.1.2). The negotiated value for this parameter returned in the response is used as the default sample interval for *MonitoredItems* assigned to this *Subscription*. The value 0 is invalid.
requestedLifetimeCount	Counter	Requested lifetime count (see 7.5 for *Counter* definition). The lifetime count shall be a mimimum of three times the keep-alive count. When the publishing timer has expired this number of times without a *Publish* request being available to send a *NotificationMessage*, then the *Subscription* shall be deleted by the *Server*.
requestedMaxKeepAlive Count	Counter	Requested maximum keep-alive count (see 7.5 for *Counter* definition). When the publishing timer has expired this number of times without requiring any *NotificationMessage* to be sent, the *Subscription* sends a keep-alive *Message* to the *Client*. The value 0 is invalid.
maxNotificationsPerPublish	Counter	The maximum number of notifications that the *Client* wishes to receive in a single *Publish* response. A value of zero indicates that there is no limit.
publishingEnabled	Boolean	A *Boolean* parameter with the following values: TRUE publishing is enabled for the *Subscription*. FALSE publishing is disabled for the *Subscription*. The value of this parameter does not affect the value of the monitoring mode *Attribute* of *MonitoredItems*.
priority	Byte	Indicates the relative priority of the *Subscription*. When more than one *Subscription* needs to send *Notifications*, the *Server* should dequeue a Publish request to the *Subscription* with the highest *priority* number. For *Subscriptions* with equal *priority* the *Server* should dequeue Publish requests in a round-robin fashion. When the keep-alive period expires for a *Subscription* it shall take precedence regardless of its *priority*, in order to prevent the *Subscription* from expiring. A Client that does not require special priority settings should set this value to zero.
Response		
responseHeader	Response Header	Common response parameters (see 7.27 for *ResponseHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* (see 7.13 for *IntegerId* definition). This identifier shall be unique for the entire *Server*, not just for the *Session*, in order to allow the *Subscription* to be transferred to another *Session* using the TransferSubscriptions service.
revisedPublishingInterval	Duration	The actual publishing interval that the *Server* will use, expressed in milliseconds. The *Server* should attempt to honor the *Client* request for this parameter, but may negotiate this value up or down to meet its own constraints.
revisedLifetimeCount	Counter	The lifetime of the *Subscription* shall be a minimum of three times the keep-alive interval negotiated by the *Server.*
revisedMaxKeepAliveCount	Counter	The actual maximum keep-alive count (see 7.5 for *Counter* definition). The *Server* should attempt to honor the *Client* request for this parameter, but may negotiate this value up or down to meet its own constraints.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.2.3 Service results

Table 84 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 84 – CreateSubscription Service Result Codes

Symbolic Id	Description
Bad_TooManySubscriptions	The *Server* has reached its maximum number of subscriptions.

5.13.3 ModifySubscription

5.13.3.1 Description

This *Service* is used to modify a *Subscription*.

5.13.3.2 Parameters

Table 85 defines the parameters for the *Service*. Changes to the publishing interval become effective the next time the publishing timer expires.

Table 85 – ModifySubscription Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* (see 7.13 for *IntegerId* definition).
requestedPublishingInterval	Duration	This interval defines the cyclic rate that the *Subscription* is being requested to return *Notifications* to the *Client*. This interval is expressed in milliseconds. This interval is represented by the publishing timer in the *Subscription* state table (see 5.13.1.2). The negotiated value for this parameter returned in the response is used as the default sample interval for *MonitoredItems* assigned to this *Subscription*. The value 0 is invalid.
requestedLifetimeCount	Counter	Requested lifetime count (see 7.5 for *Counter* definition). The lifetime count shall be a mimimum of three times the keep keep-alive count. When the publishing timer has expired this number of times without a *Publish* request being available to send a *NotificationMessage*, then the *Subscription* shall be deleted by the *Server*.
requestedMaxKeepAliveCount	Counter	Requested maximum keep-alive count (see 7.5 for *Counter* definition). When the publishing timer has expired this number of times without requiring any *NotificationMessage* to be sent, the *Subscription* sends a keep-alive *Message* to the *Client*. The value 0 is invalid.
maxNotificationsPerPublish	Counter	The maximum number of notifications that the *Client* wishes to receive in a single *Publish* response. A value of zero indicates that there is no limit.
priority	Byte	Indicates the relative priority of the *Subscription*. When more than one *Subscription* needs to send *Notifications*, the *Server* should dequeue a Publish request to the *Subscription* with the highest *priority* number. For *Subscriptions* with equal *priority* the *Server* should dequeue Publish requests in a round-robin fashion. Any *Subscription* that needs to send a keep-alive *Message* shall take precedence regardless of its *priority*, in order to prevent the *Subscription* from expiring. A Client that does not require special priority settings should set this value to zero.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
revisedPublishingInterval	Duration	The actual publishing interval that the *Server* will use, expressed in milliseconds. The *Server* should attempt to honor the *Client* request for this parameter, but may negotiate this value up or down to meet its own constraints.
revisedLifetimeCount	Counter	The lifetime of the *Subscription* shall be a minimum of three times the keep-alive interval negotiated by the *Server.*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Name	Type	Description
Request		
revisedMaxKeepAliveCount	Counter	The actual maximum keep-alive count (see 7.5 for *Counter* definition). The *Server* should attempt to honor the *Client* request for this parameter, but may negotiate this value up or down to meet its own constraints.

5.13.3.3 Service results

Table 86 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 86 – ModifySubscription Service Result Codes

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

5.13.4 SetPublishingMode

5.13.4.1 Description

This *Service* is used to enable sending of *Notifications* on one or more *Subscriptions*.

5.13.4.2 Parameters

Table 87 defines the parameters for the *Service*.

Table 87 – SetPublishingMode Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
publishingEnabled	Boolean	A *Boolean* parameter with the following values: TRUE publishing of *NotificationMessages* is enabled for the *Subscription*. FALSE publishing of *NotificationMessages* is disabled for the *Subscription*. The value of this parameter does not affect the value of the monitoring mode *Attribute* of *MonitoredItems*. Setting this value to FALSE does not discontinue the sending of keep-alive *Messages*.
subscriptionIds []	IntegerId	List of *Server*-assigned identifiers for the *Subscriptions* to enable or disable (see 7.13 for *IntegerId* definition).
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *Subscriptions* to enable/disable (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *subscriptionIds* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *Subscriptions* to enable/disable (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *subscriptionIds* 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.

5.13.4.3 Service results

Table 88 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 88 – SetPublishingMode 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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.4.4 StatusCodes

Table 89 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 89 – SetPublishingMode Operation Level Result Codes

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

5.13.5 Publish

5.13.5.1 Description

This *Service* is used for two purposes. First, it is used to acknowledge the receipt of *NotificationMessages* for one or more *Subscriptions*. Second, it is used to request the *Server* to return a *NotificationMessage* or a keep-alive *Message*. Since *Publish* requests are not directed to a specific *Subscription*, they may be used by any *Subscription*. Subclause 5.13.1.2 describes the use of the *Publish Service*.

Client strategies for issuing *Publish* requests may vary depending on the networking delays between the *Client* and the *Server*. In many cases, the *Client* may wish to issue a *Publish* request immediately after creating a *Subscription*, and thereafter, immediately after receiving a *Publish* response.

In other cases, especially in high latency networks, the *Client* may wish to pipeline *Publish* requests to ensure cyclic reporting from the *Server*. Pipelining involves sending more than one *Publish* request for each *Subscription* before receiving a response. For example, if the network introduces a delay between the *Client* and the *Server* of 5 s and the publishing interval for a *Subscription* is 1 s, then the *Client* will have to issue *Publish* requests every second instead of waiting for a response to be received before sending the next request.

A server should limit the number of active *Publish* requests to avoid an infinite number since it is expected that the *Publish* requests are queued in the *Server*. But a Server shall accept more queued *Publish* requests than created Subscriptions. It is expected that a *Server* supports several *Publish* requests per *Subscription*. When a *Server* receives a new *Publish* request that exceeds its limit it shall dequeue the oldest *Publish* request and return a response with the result set to *Bad_TooManyPublishRequests*. If a *Client* receives this *Service* result for a *Publish* request it shall not issue another *Publish* request before one of its outstanding *Publish* requests is returned from the Server.

Clients can limit the size of *Publish* reponses with the *maxNotificationsPerPublish* parameter passed to the *CreateSubscription Service*. However, this could still result in a message that is too large for the *Client* or *Server* to process. In this situation, the *Client* will find that either the *SecureChannel* goes into a fault state and needs to be re-established or the *Publish* response returns an error and calling the *Republish Service* also returns an error. If either situation occurs then the *Client* will have to adjust its message processing limits or the parameters for the *Subscription and/or MonitoredItems*.

The return diagnostic info setting in the request header of the *CreateMonitoredItems* or the last *ModifyMonitoredItems Service* is applied to the *Monitored Items* and is used as the diagnostic information settings when sending Notifications in the *Publish* response.

5.13.5.2 Parameters

Table 90 defines the parameters for the *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 90 – Publish Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscription Acknowledgements []	Subscription Acknowledgement	The list of acknowledgements for one or more *Subscriptions*. This list may contain multiple acknowledgements for the same *Subscription* (multiple entries with the same *subscriptionId*).
subscriptionId	IntegerId	The *Server* assigned identifier for a *Subscription* (see 7.13 for *IntegerId* definition).
sequenceNumber	Counter	The sequence number being acknowledged (see 7.5 for *Counter* definition). The *Server* may delete the *Message* with this sequence number from its retransmission queue.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
subscriptionId	IntegerId	The *Server*-assigned identifier for the *Subscription* for which *Notifications* are being returned (see 7.13 for *IntegerId* definition). The value 0 is used to indicate that there were no *Subscriptions* defined for which a response could be sent.
availableSequence Numbers []	Counter	A list of sequence number ranges that identify unacknowledged *NotificationMessages* that are available for retransmission from the *Subscription*'s retransmission queue. This list is prepared after processing the acknowledgements in the request (see 7.5 for *Counter* definition).
moreNotifications	Boolean	A *Boolean* parameter with the following values: TRUE the number of *Notifications* that were ready to be sent could not be sent in a single response. FALSE all *Notifications* that were ready are included in the response.
notificationMessage	Notification Message	The *NotificationMessage* that contains the list of *Notifications*. The *NotificationMessage* parameter type is specified in 7.20.
results []	StatusCode	List of results for the acknowledgements (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *subscriptionAcknowledgements* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the acknowledgements (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *subscriptionAcknowledgements* 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.

5.13.5.3 Service results

Table 91 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 91 – Publish Service Result Codes

Symbolic Id	Description
Bad_TooManyPublishRequests	The server has reached the maximum number of queued publish requests.
Bad_NoSubscription	There is no subscription available for this session.

5.13.5.4 StatusCodes

Table 92 defines values for the *acknowledgeResults* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 92 – Publish Operation Level Result Codes

Symbolic Id	Description
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.
Bad_SequenceNumberUnknown	The sequence number is unknown to the server.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.6 Republish

5.13.6.1 Description

This *Service* requests the *Subscription* to republish a *NotificationMessage* from its retransmission queue. If the *Server* does not have the requested *Message* in its retransmission queue, it returns an error response.

See 5.13.1.2 for the detail description of the behaviour of this *Service*.

5.13.6.2 Parameters

Table 93 defines the parameters for the *Service*.

Table 93 – Republish Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionId	IntegerId	The *Server* assigned identifier for the *Subscription* to be republished (see 7.13 for *IntegerId* definition).
retransmitSequence Number	Counter	The sequence number of a specific *NotificationMessage* to be republished (see 7.5 for *Counter* definition).
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
notificationMessage	Notification Message	The requested *NotificationMessage*. The *NotificationMessage* parameter type is specified in 7.18.

5.13.6.3 Service results

Table 94 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 94 – Republish Service Result Codes

Symbolic Id	Description
Bad_SubscriptionIdInvalid	See Table 165 for the description of this result code.
Bad_MessageNotAvailable	The requested message is no longer available.

5.13.7 TransferSubscriptions

5.13.7.1 Description

This *Service* is used to transfer a *Subscription* and its *MonitoredItems* from one *Session* to another. For example, a *Client* may need to reopen a *Session* and then transfer its *Subscriptions* to that *Session*. It may also be used by one *Client* to take over a *Subscription* from another *Client* by transferring the *Subscription* to its *Session*.

The *authenticationToken* contained in the request header identifies the *Session* to which the *Subscription* and *MonitoredItems* shall be transferred. The *Server* shall validate that the *Client* of that *Session* is operating on behalf of the same user and that the potentially new *Client* supports the *Profiles* that are necessary for the *Subscription*. If the *Server* transfers the *Subscription*, it returns the sequence numbers of the *NotificationMessages* that are available for retransmission. The *Client* should acknowledge all *Messages* in this list for which it will not request retransmission.

If the *Server* transfers the *Subscription* to the new *Session*, the *Server* shall issue a *StatusChangeNotification notificationMessage* with the status code Good_SubscriptionTransferred. The *StatusChangeNotification notificationMessage* type is defined in 7.19.4.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.7.2 Parameters

Table 95 defines the parameters for the *Service*.

Table 95 – TransferSubscriptions Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionIds []	IntegerId	List of identifiers for the *Subscriptions* to be transferred to the new *Client* (see 7.13 for *IntegerId* definition). These identifiers are transferred from the primary *Client* to a backup *Client* via external mechanisms.
sendInitialValues	Boolean	A *Boolean* parameter with the following values: TRUE the first Publish response after the TransferSubscriptions call shall contain the current values of all Monitored Items in the Subscription where the Monitoring Mode is set to Reporting. FALSE the first Publish response after the TransferSubscriptions call shall contain only the value changes since the last Publish response was sent. This parameter only applies to MonitoredItems used for monitoring Attribute changes.
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	TransferResult	List of results for the *Subscriptions* to transfer. The size and order of the list matches the size and order of the *subscriptionIds* request parameter.
statusCode	StatusCode	*StatusCode* for each *Subscription* to be transferred (see 7.33 for *StatusCode* definition).
availableSequence NumbersRanges []	Counter	A list of sequence number ranges that identify *NotificationMessages* that are in the *Subscription*'s retransmission queue. This parameter is null if the transfer of the *Subscription* failed. The *NumericRange* type is defined in 7.21.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *Subscriptions* to transfer (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *subscriptionIds* 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.

5.13.7.3 Service results

Table 96 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 96 – TransferSubscriptions 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.
Bad_UserAccessDenied	See Table 165 for the description of this result code. The *Client* of the current *Session* is not operating on behalf of the same user as the *Session* that owns the *Subscription*.
Bad_InsufficientClientProfile	The *Client* of the current *Session* does not support one or more *Profiles* that are necessary for the *Subscription*.

5.13.7.4 StatusCodes

Table 97 defines values for the operation level *statusCode* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 97 – TransferSubscriptions Operation Level Result Codes

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.8 DeleteSubscriptions

5.13.8.1 Description

This *Service* is invoked by the *Client* to delete one or more *Subscriptions* that it has created and that have not been transferred to another *Client* or that have been transferred to it.

Successful completion of this *Service* causes all *MonitoredItems* that use the *Subscription* to be deleted. If this is the last *Subscription* assigned to the *Client* issuing the request, then all *Publish* requests queued by that *Client* are dequeued and a negative response is returned for each.

5.13.8.2 Parameters

Table 98 defines the parameters for the *Service*.

Table 98 – DeleteSubscriptions Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.26 for *RequestHeader* definition).
subscriptionIds []	IntegerId	The *Server*-assigned identifier for the *Subscription* (see 7.13 for *IntegerId* definition).
Response		
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).
results []	StatusCode	List of *StatusCodes* for the *Subscriptions* to delete (see 7.33 for *StatusCode* definition). The size and order of the list matches the size and order of the *subscriptionIds* request parameter.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information for the *Subscriptions* to delete (see 7.8 for *DiagnosticInfo* definition). The size and order of the list matches the size and order of the *subscriptionIds* 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.

5.13.8.3 Service results

Table 99 defines the *Service* results specific to this *Service*. Common *StatusCodes* are defined in Table 165.

Table 99 – DeleteSubscriptions 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.

5.13.8.4 StatusCodes

Table 100 defines values for the *results* parameter that are specific to this *Service*. Common *StatusCodes* are defined in Table 166.

Table 100 – DeleteSubscriptions Operation Level Result Codes

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6 Service behaviours

6.1 Security

6.1.1 Overview

The OPC UA *Services* define a number of mechanisms to meet the security requirements outlined in IEC/TR 62541-2. This subclause describes a number of important security-related procedures that *OPC UA Applications* shall follow.

6.1.2 Obtaining and Installing an Application Instance Certificate

All *OPC UA Applications* require an application instance certificate which shall contain the following information:

- the network name or address of the computer where the application runs;
- the name of the organisation that administers or owns the application;
- the name of the application;
- the uri of the application instance;
- the name of the certificate authority that issued the certificate;
- the issue and expiry date for the certificate;
- the public key issued to the application by the certificate authority (CA);
- a digital signature created by the certificate authority (CA).

In addition, each application instance certificate has a private key which should be stored in a location that can only be accessed by the application. If this private key is compromised, the administrator shall assign a new application instance certificate and private key to the application.

This certificate may be generated automatically when the application is installed. In this situation the private key assigned to the certificate shall be used to create the certificate signature. Certificates created in this way are called self-signed certificates.

If the administrator responsible for the application decides that a self-signed certificate does not meet the security requirements of the organisation, then the administrator should install a certificate issued by a certification authority. The steps involved in requesting an application instance certificate from a certificate authority are shown in Figure 18.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

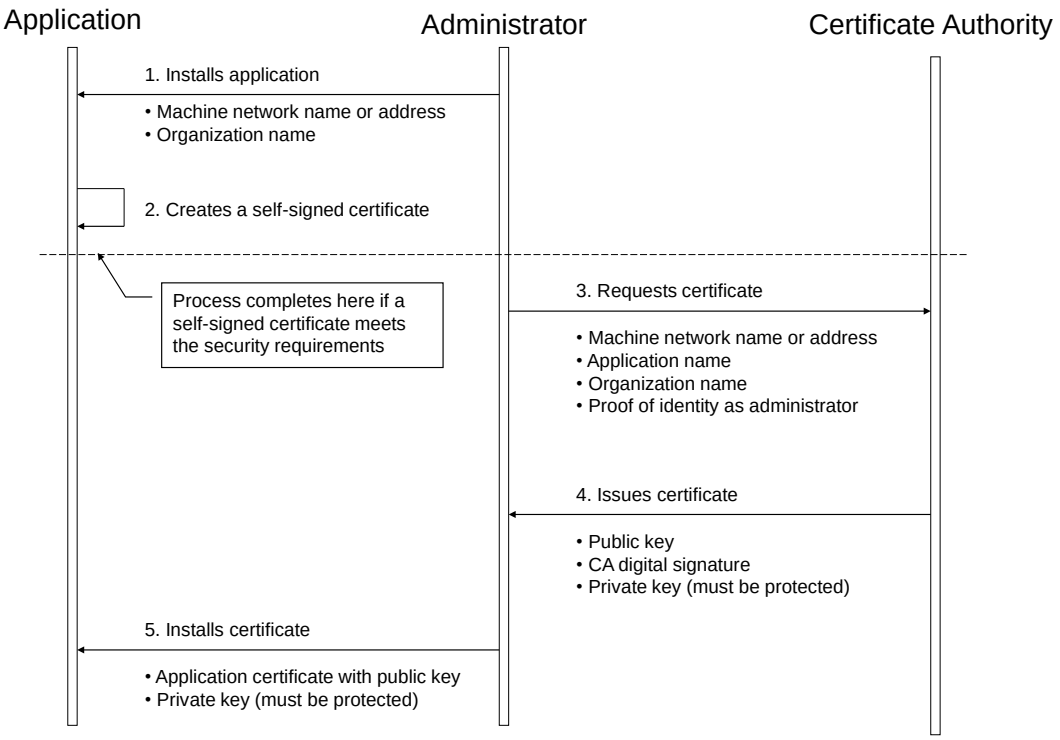


Figure 18 – Obtaining and installing an Application Instance Certificate

Figure 18 illustrates the interactions between the *Application*, the *Administrator* and the *CertificateAuthority*. The *Application* is as *OPC UA Application* installed on a single machine. The *Administrator* is the person responsible for managing the machine and the *OPC UA Application*. The *CertificateAuthority* is an entity that can issue digital certificates that meet the requirements of the organisation deploying the *OPC UA Application*.

If the *Administrator* decides that a self-signed certificate meets the security requirements for the organisation, then the *Administrator* may skip Steps 3 through 5. Application vendors shall always create a default self-signed certificate during the installation process. Every *OPC UA Application* shall allow the *Administrators* to replace application instance certificates with certificates that meet their requirements.

When the *Administrator* requests a new certificate from a certificate authority, the certificate authority may require that the *Administrator* provide proof of authorization to request certificates for the organisation that will own the certificate. The exact mechanism used to provide this proof depends on the certificate authority.

Vendors may choose to automate the process of acquiring certificates from an authority. If this is the case, the *Administrator* would still go through the steps illustrated in Figure 18, however, the installation program for the application would do them automatically and only prompt the *Administrator* to provide information about the application instance being installed.

6.1.3 Obtaining and installing a Software Certificate

All *OPC UA Applications* may have one or more software certificates that are issued by certification authorities and specify the profiles that the application supports. These software certificates contain the following information:

- the name of the vendor responsible for the product;
- the name of the product;
- the uri for the product;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- the software version and build number;
- the vendor product certificate;
- a list of profiles supported by the application;
- the certification testing status for each supported profile;
- the name of the testing authority that issued the certificate;
- the issue and expiry date for the certificate;
- the public key issued to the application by the certifying authority;
- a digital signature created by the certifying authority.

The product vendor is responsible for completing the certification process and requesting software certificates from the certification authorities. The vendor shall install the software certificates when an application is installed on machine. When distributing these certificates, the application vendors should take precautions to prevent unauthorized users from acquiring their software certificates and using them for applications that the vendor did not develop. Misused software certificates are not a security risk, but vendors could find that they are blamed for interoperability problems caused by certificates use by unauthorized applications.

Vendors may choose to assign their own *Certificate* to the application (called the Vendor Product Certificate). This Certificate would then be incorporated in the *Software Certificate*. The private key for the vendor product certificate is managed by the Vendor.

The steps involved in acquiring and installing a software certificate from a certificate authority are shown in Figure 19.

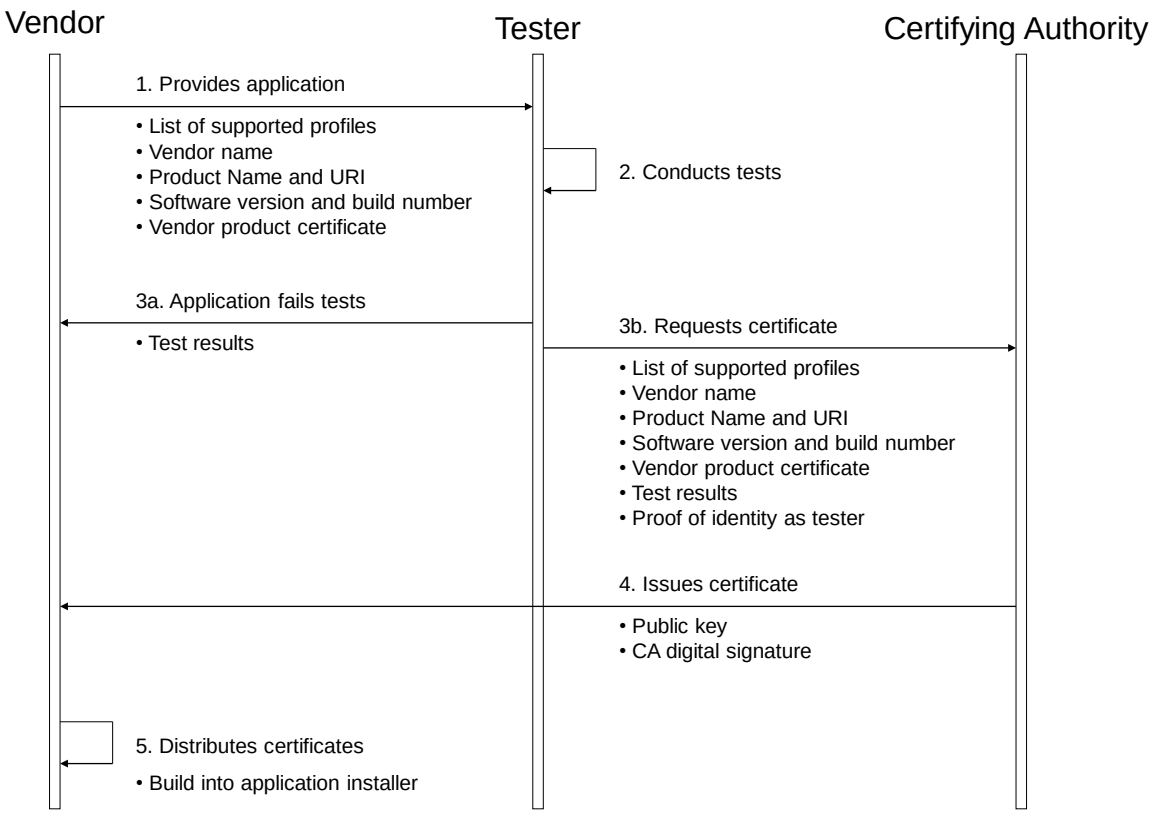


Figure 19 – Obtaining and Installing a Software Certificate

Figure 19 illustrates the interactions between the *Vendor*, the *Tester* and the *CertifyingAuthority*. The *Vendor* is the organisation responsible for the *OPC UA Application*. The *Tester* may be the vendor (for self-certification) or it may be a third-party testing facility. The *CertifyingAuthority* is a PKI certificate authority managed by the organisation that created the *OPC UA Application* profiles and certification programmes.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The *CertifyingAuthority* will issue certificates only to people it trusts. For that reason, the *Tester* shall provide proof of identity before the *CertifyingAuthority* will issue a certificate.

6.1.4 Determining if a Certificate is Trusted

Applications shall never communicate with another application that they do not trust. An *Application* decides if another application is trusted by checking whether the application instance *Certificate* for the other application is trusted. Applications shall rely on lists of Certificates provided by the Administrator to determine trust. There are two separate lists: a list of trusted Applications and a list of trusted Certificate Authorities (CAs). If an application is not directly trusted (i.e. its Certificate is not in the list of trusted Applications) then the Application shall build a chain of Certificates back to a trusted CA.

When building a chain each Certificate in the chain shall be validated. If any validation error occurs then the trust check fails. Some validation errors are non-critical which means they can be suppressed by a user of an Application with the appropriate privileges. Suppressed validation errors are always reported via auditing (i.e. an appropriate Audit event is raised).

Building a trust chain requires access to all Certificates in the chain. These Certificates may be stored locally or they may be provided with the application Certificate. Processing stops if a CA certificate cannot be found or if it is in the list of trusted CAs.

Table 101 specifies the steps used to validate a Certificate in the order that they should be followed. These steps are repeated for each certificate in the chain until a trusted certificate is found. Each validation step has a unique error status and audit event type that shall be reported if the check fails. The audit event is in addition to any audit event that was generated for the particular *Service* that was invoked. The *Service* audit event in its message text shall include the audit eventID of the *AuditCertifcateEvent* (for more details, see 6.2). Processing halts if an error occurs, unless it is non-critical and it has been suppressed.

Application instance certificates shall not be used in a *Client* or *Server* until they have been evaluated and marked as trusted. This can happen automatically by a PKI trust chain or in an offline manner where the *Certificate* is marked as trusted by an administrator after evaluation.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 101 – Certificate Validation Steps

Step	Error/AuditEvent	Description
Certificate Structure	Bad_SecurityChecksFailed AuditCertificateInvalidEventType	The certificate structure is verified. This error may not be suppressed.
Validity Period	Bad_CertificateTimeInvalid Bad_CertificateIssuerTimeInvalid AuditCertificateExpiredEventType	The current time shall be after the start of the validity period and before the end. This error may be suppressed.
Host Name	Bad_CertificateHostNameInvalid AuditCertificateDataMismatchEventType	The HostName in the URL used to connect to the Server shall be the same as one of the HostNames specified in the Certificate. This error may be suppressed.
URI	Bad_CertificateUriInvalid AuditCertificateDataMismatchEventType	Application and Software Certificates contain an application or product URI that shall match the URI specified in the *ApplicationDescription* provided with the Certificate. This check is skipped for CA Certificates. This error may not be suppressed. The *gatewayServerUri* is used to validate an *Application Certificate* when connecting to a *Gateway Server* (See 7.1).
Certificate Usage	Bad_CertificateUseNotAllowed Bad_CertificateIssuerUseNotAllowed AuditCertificateMismatchEventType	Each Certificate has a set of uses for the Certificate. These uses shall match use requested for the Certificate (i.e. Application, Software or CA). This error may be suppressed unless the Certificate indicates that the usage is mandatory.
Trust List Check	None	No further checks are required if the Certificate is in the Trust List. The Administrator shall completely validate any Certificate before placing it in the Trust List.
Find Issuer Certificate	Bad_CertificateUntrusted AuditCertificateUntrustedEventType	A Certificate cannot be trusted if the Issuer Certificate is unknown. A self-signed Certificate is its own issuer.
Signature	Bad_SecurityChecksFailed AuditCertificateInvalidEventType	A Certificate with an invalid signature shall always be rejected.
Find Revocation List	Bad_CertificateRevocationUnknown Bad_CertificateIssuerRevocationUnknown AuditCertificateRevokedEventType	Each CA Certificate may have a revocation list. This check fails if this list is not available (i.e. a network interruption prevents the Application from accessing the list). No error is reported if the Administrator disables revocation checks for a CA Certificate. This error may be suppressed.
Revocation Check	Bad_CertificateRevoked Bad_CertificateIssuerRevoked AuditCertificateRevokedEventType	The Certificate has been revoked and may not be used. This error may not be suppressed.

Certificates are usually placed in a central location called a CertificateStore. Figure 20 illustrates the interactions between the *Application*, the *Administrator* and the *CertificateStore*. The *CertificateStore* could be on the local machine or in some central server. The exact mechanisms used to access the *CertificateStore* depend on the application and PKI environment set up by the *Administrator*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

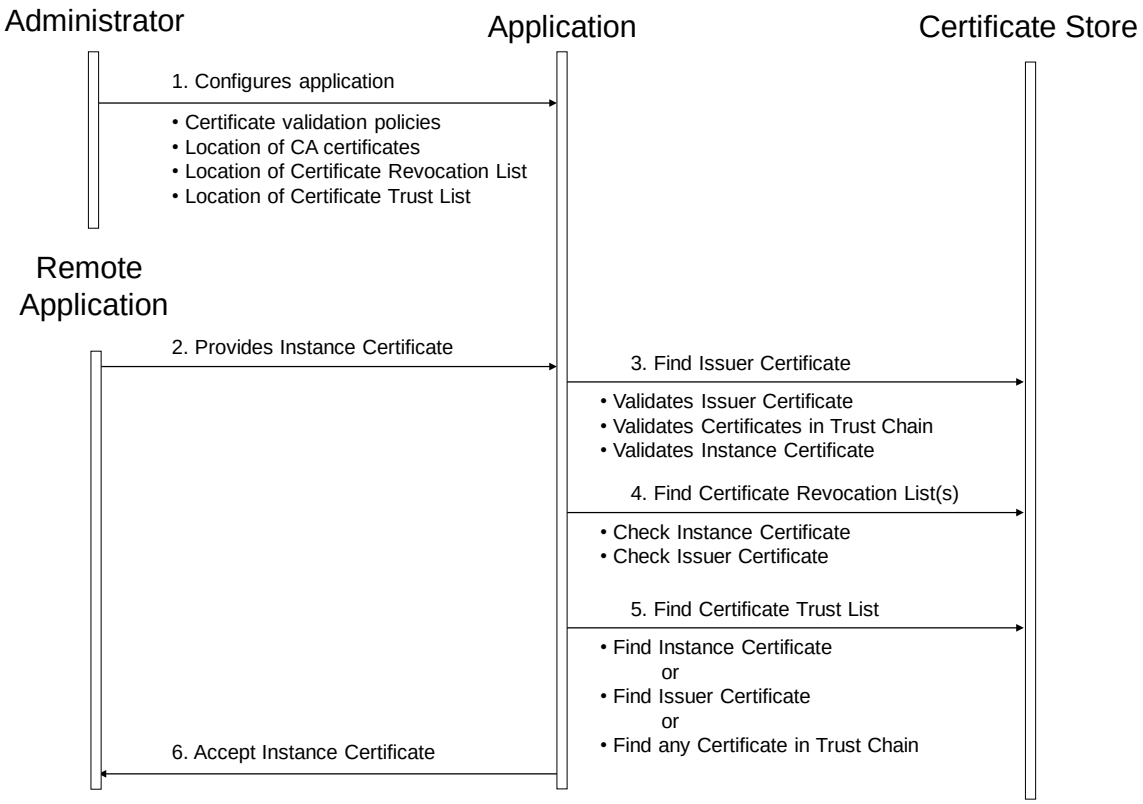


Figure 20 – Determining if a Application Instance Certificate is Trusted

6.1.5 Validating a Software Certificate

OPC UA Applications shall validate the *SoftwareCertificates* provided by the applications that they communicate with.

A *SoftwareCertificate* is valid if:

- the signature on the *SoftwareCertificate* is valid;
- the *SoftwareCertificate* has passed its issue date and it has not expired;
- the *SoftwareCertificate* has not been revoked by the issuer;
- the issuer *Certificate* is valid and has not been revoked by the CA that issued it.

The steps used to validate *SoftwareCertificates* are almost the same as the steps described for application instance *Certificates* in 6.1.4. The only difference is *SoftwareCertificates* are not checked against the *TrustList* for the *Application.*

SoftwareCertificates always contain an application *Certificate* which is owned by the vendor that distributes the application. The application *Certificate* may also be in the *Certificate* chain used to issue application instance *Certificates*. For this reason, *OPC UA Applications* may reject *SoftwareCertificates* provided by applications if the application *Certificate* is not part of the *Certificate* chain for the application instance *Certificate*. *OPC UA Applications* should allow *Administrators* to require this behaviour. Profiles defined in IEC 62541-7 may further specify expected *Certificate* handling behaviors.

6.1.6 Creating a SecureChannel

All *OPC UA Applications* shall establish a *SecureChannel* before creating a *Session*. This *SecureChannel* requires that both applications have access to certificates that can be used to

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

encrypt and sign *Messages* exchange. The application instance certificates installed by following the process described in 6.1.2 may be used for this purpose.

The steps involved in establishing a *SecureChannel* are shown in Figure 21.

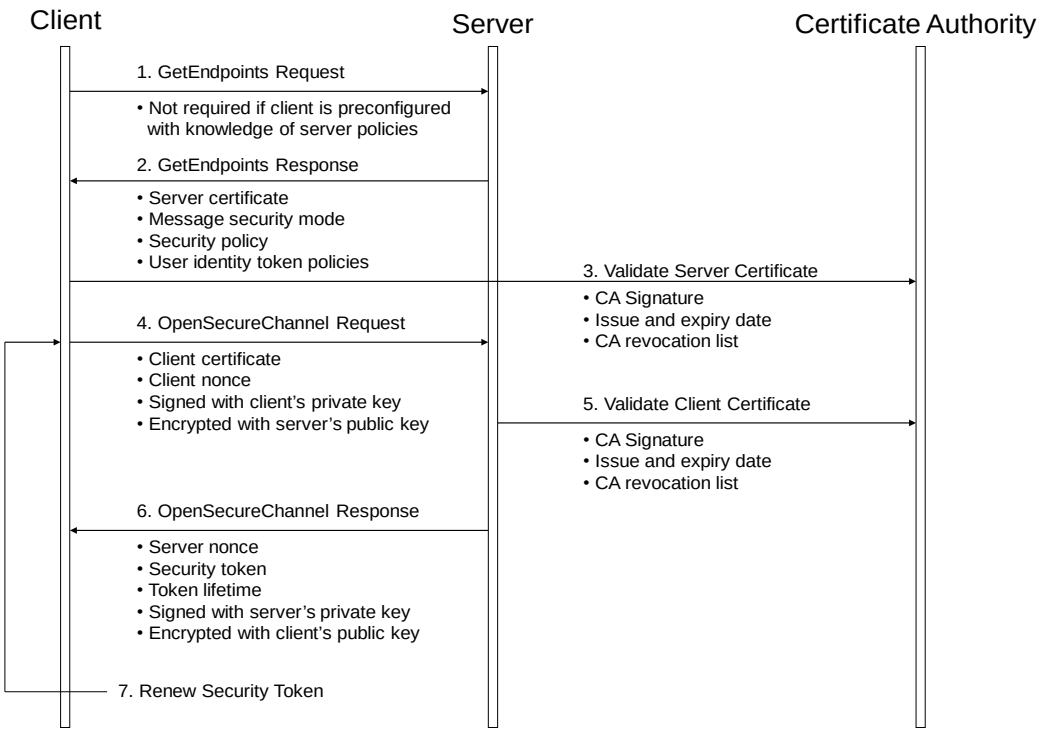


Figure 21 – Establishing a SecureChannel

Figure 21 above assumes *Client* and *Server* have online access to a certificate authority (CA). If online access is not available and if the administrator has installed the CA public key on the local machine, then the *Client* and *Server* shall still validate the application certificates using that key. The figure shows only one CA, however, there is no requirement that the *Client* and *Server* certificates be issued by the same authority. A self-signed application instance certificate does not need to be verified with a CA. Any certificate shall be rejected if it is not in a trust list provided by the administrator.

Both the *Client* and *Server* shall have a list of certificates that they have been configured to trust (sometimes called the Certificate Trust List or CTL). These trusted certificates may be certificates for certificate authorities or they may be *OPC UA Application* instance certificates. *OPC UA Applications* shall be configured to reject connections with applications that do not have a trusted certificate.

Certificates can be compromised, which means they should no longer be trusted. Administrators can revoke a certificate by removing it from the trust list for all applications or the CA can add the certificate to the Certificate Revocation List (CRL) for the issuer *Certificate*. Administrators may save a local copy of the CRL for each issuer *Certificate* when online access is not available.

A *Client* does not need to call *GetEndpoints* each time it connects to the *Server*. This information should change rarely and the *Client* can cache it locally. If the *Server* rejects the *OpenSecureChannel* request the *Client* should call *GetEndpoints* and make sure the *Server* configuration has not changed.

There are two security risks which a *Client* shall be aware of when using the *GetEndpoints Service*. The first could come from a rogue *Discovery Server* that tries to direct the *Client* to a

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

rogue *Server*. For this reason the *Client* shall verify that the *ServerCertificate* in the *EndpointDescription* is a trusted *Certificate* before it calls *CreateSession*.

The second security risk comes from third party that alters the contents of the *EndpointDescriptions* as they are transferred over the network back to the *Client*. The Client protects itself against this by comparing the list of *EndpointDescriptions* returned from the *GetEndpoints Service* with the list returned in the *CreateSession* response.

The exact mechanisms for using the security token to sign and encrypt *Messages* exchanged over the *SecureChannel* are described in IEC 62541-6. The process for renewing tokens is also described in detail in IEC 62541-6.

In many cases, the certificates used to establish the *SecureChannel* will be the application instance certificates. However, some *Communication Stacks* might not support certificates that are specific to a single application. Instead, they expect all communication to be secured with a certificate specific to a user or the entire machine. For this reason, *OPC UA Applications* will need to exchange their application instance certificates when creating a *Session*.

6.1.7 Creating a Session

Once an OPC UA *Client* has established a *SecureChannel* with a *Server* it can create an OPC UA *Session*.

The steps involved in establishing a *Session* are shown in Figure 22.

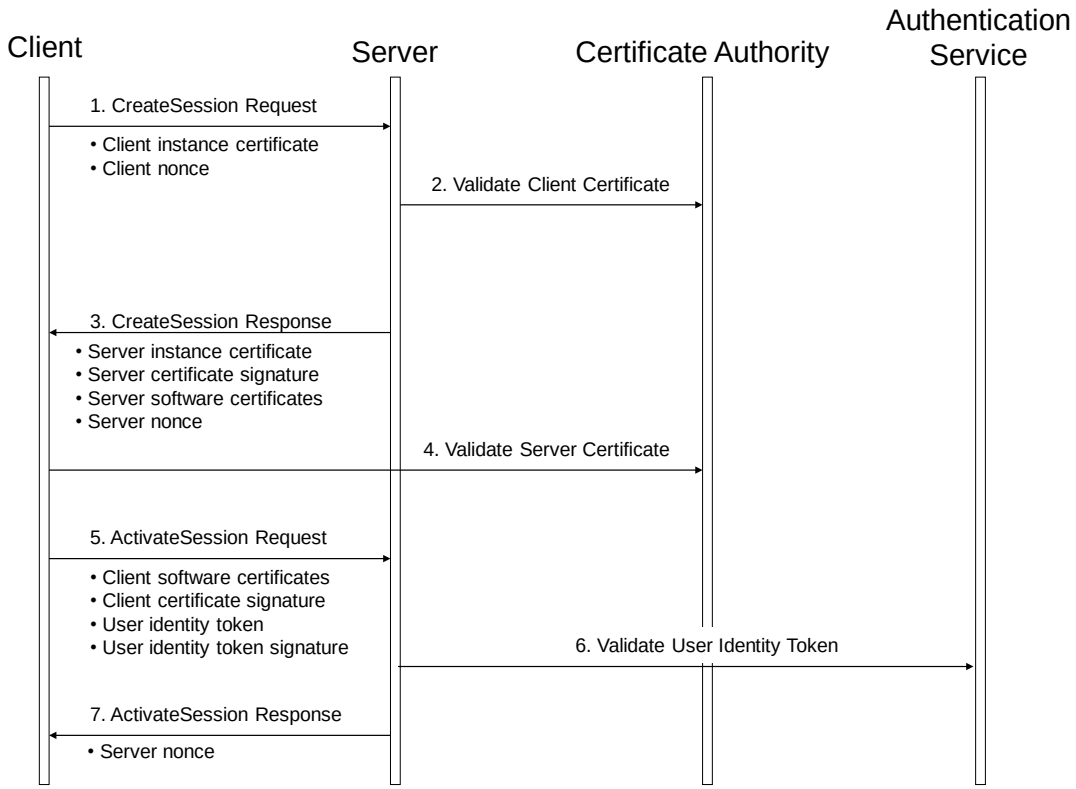


Figure 22 – Establishing a Session

Figure 22 above illustrates the interactions between a *Client*, a *Server*, a certificate authority (CA) and an authentication service. The CA is responsible for issuing the application certificates. If the *Client* or *Server* do not have online access to the CA, then they shall validate the application certificates using the CA public key that the administrator shall install on the local machine.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The authentication service is a central database that can verify that user token provided by the *Client*. This authentication service may also tell the *Server* what access rights the user has. The authentication service depends on the user identity token. It could be a certificate authority, a Kerberos ticket granting service, a WS-Trust *Server* or a proprietary database of some sort.

The *Client* and *Server* shall prove possession of their application certificates by signing the certificates with a nonce appended. The exact mechanism used to create the proof of possession signatures is described in 5.6.2. Similarly, the *Client* shall prove possession of some types of user identity tokens by creating signatures with the secret associated with the token.

6.1.8 Impersonating a User

Once an OPC UA *Client* has established a *Session* with a *Server* it can change the user identity associated with the *Session* by calling the *ActivateSession* service.

The steps involved in impersonating a user are shown in Figure 23.

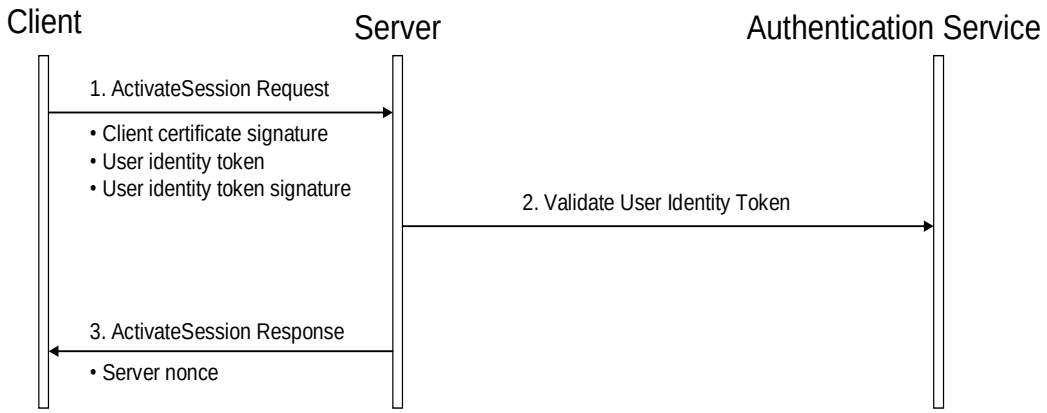


Figure 23 – Impersonating a User

6.2 Auditing

6.2.1 Overview

Auditing is a requirement in many systems. It provides a means of tracking activities that occur as part of normal operation of the system. It also provides a means of tracking abnormal behaviour. It is also a requirement from a security standpoint. For more information on the security aspects of auditing, see IEC/TR 62541-2. This subclause describes what is expected of an OPC UA *Server* and *Client* with respect to auditing and it details the audit requirements for each service set. Auditing can be accomplished using one or both of the following methods.

a) The *OPC UA Application* that generates the audit event can log the audit entry in a log file or other storage location.

b) The OPC UA *Server* that generates the audit event can publish the audit event using the OPC UA event mechanism. This allows an external OPC UA *Client* to subscribe to and log the audit entries to a log file or other storage location.

6.2.2 General audit logs

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

create a local audit log entry for a request that it submits. This parameter allows this *Client* to pass the identifier for this entry with the request. If this *Server* also maintains an audit log, it should include this id in its audit log entry that it writes. When this log is examined and that 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.

6.2.3 General audit Events

A *Server* that maintains an audit log shall provide the audit log entries via *Event Messages*. The *AuditEventType* and its sub-types are defined in IEC 62541-3. An audit *Event Message* also includes the audit record Id. The details of the *AuditEventType* and its subtypes are defined in IEC 62541-5. A *Server* that is an aggregating *Server* that supports auditing shall also subscribe for audit events for all of the *Servers* that it is aggregating (assuming they provide auditing). The combined stream should be available from the aggregating *Server*.

6.2.4 Auditing for Discovery Service Set

This *Service Set* can be separated into two groups: Services that are called by OPC UA *Clients* and *Services* that are invoked by OPC UA *Servers*. The *FindServers* and *GetEndpoints Services* that are called by OPC UA *Clients* may generate audit entries for failed *Service* invocations. The *RegisterServer* Service that is invoked by OPC UA *Servers* shall generate audit entries for all new registrations and for failed *Service* invocations. These audit entries shall include the *Server* URI, *Server* names, *Discovery* URIs and isOnline status. Audit entries should not be generated for *RegisterServer* invocation that does not cause changes to the registered *Servers*.

6.2.5 Auditing for SecureChannel Service Set

All *Services* in this *Service Set* for *Servers* that support auditing may generate audit entries and shall generate audit *Events* for failed service invocations and for successful invocation of the *OpenSecureChannel* and *CloseSecureChannel* Services. The *Client* generated audit entries should be setup prior to the actual call, allowing the correct audit record Id to be provided. The *OpenSecureChannel* Service shall generate an audit *Event* of type *AuditOpenSecureChannelEventType* or a subtype of it. The *CloseSecureChannel* service shall generate an audit *Event* of type *AuditCloseSecureChannelEventType* or a subtype of it. Both of these *Event* types are subtypes of the *AuditChannelEventType*. See IEC 62541-5 for the detailed assignment of the *SourceNode*, the *SourceName* and additional parameters. For the failure cases the *Message* for *Events* of this type should include a description of why the service failed. This description should be more detailed than what was returned to the client. From a security point of view a client only needs to know that it failed, but from an *Auditing* point of view the exact details of the failure need to be known. In the case of *Certificate* validation errors the description should include the *AuditEventId* of the specific *AuditCertificateEvent* that was generated to report the certificate error. The *AuditCertifcateEvent* shall also contain the detailed certificate validation error. The additional parameters should include the details of the request. It is understood that these events may be generated by the underlying *Communication Stacks* in many cases, but they shall be made available to the *Server* and the *Server* shall report them.

6.2.6 Auditing for Session Service Set

All *Services* in this *Service Set* for *Servers* that support auditing may generate audit entries and shall generate audit *Events* for both successful and failed *Service* invocations. These *Services* shall generate an audit *Event* of type *AuditSessionEventType* or a subtype of it. In particular, they shall generate the base *EventType* or the appropriate subtype, depending on the service that was invoked. The *CreateSession* service shall generate *AuditCreateSessionEventType* events or sub-types of it. The *ActivateSession* service shall generate *AuditActivateSessionType* events or sub-types of it. When the *ActivateSession Service* is called to change the user identity then the server shall generate *AuditImpersonateUserEventType* events or subtypes of it. The CloseSession service shall generate the base *EventType* of *AuditSessionEventType* or subtypes of it. See IEC 62541-5 for the detailed assignment of the *SourceNode*, the *SourceName* and additional parameters.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

For the failure case the *Message* for *Events* of this type should include a description of why the *Service* failed. The additional parameters should include the details of the request.

This *Service Set* shall also generate additional audit events in the cases when *Certificate* validation errors occur. These audit events are generated in addition to the *AuditSessionEvents*.

For *Clients*, that support auditing, accessing the services in the *Session Service Set* shall generate audit entries for both successful and failed invocations of the *Service*. These audit entries should be setup prior to the actual *Service* invocation, allowing the invocation to contain the correct audit record id.

6.2.7 Auditing for NodeManagement Service Set

All Services in this *Service Set* for *Servers* that support auditing may generate audit entries and shall generate audit *Events* for both successful and failed *Service* invocations. These *Services* shall generate an audit *Event* of type *AuditNodeManagementEventType* or subtypes of it. See IEC 62541-5 for the detailed assignment of the *SourceNode*, the *SourceName* and additional parameters. For the failure case, the *Message* for *Events* of this type should include a description of why the service failed. The additional parameters should include the details of the request.

For *Clients* that support auditing, accessing the *Services* in the *NodeManagement Service Set* shall generate audit entries for both successful and failed invocations of the *Service*. All audit entries should be setup prior to the actual *Service* invocation, allowing the invocation to contain the correct audit record id.

6.2.8 Auditing for Attribute Service Set

The *Write* or *HistoryUpdate* Services in this *Service Set* for *Servers* that support auditing may generate audit entries and shall generate audit *Events* for both successful and failed *Service* invocations. These *Services* shall generate an audit *Event* of type *AuditUpdateEventType* or subtypes of it. In particular, the *Write Service* shall generate an audit event of type *AuditWriteUpdateEventType* or a subtype of it. The *HistoryUpdate Service* shall generate an audit *Event* of type *AuditHistoryUpdateEvent* or a subtype of it. Three subtypes of *AuditHistoryUpdateEvent* are defined as *AuditHistoryEventUpdateEventType*, *AuditHistoryValueUpdateEventType* and *AuditHistoryDeleteEventType*. The subtype depends on the type of operation being performed, historical event update, historical data value update or a historical delete. See IEC 62541-5 for the detailed assignment of the *SourceNode*, the *SourceName* and additional parameters. For the failure case the *Message* for *Events* of this type should include a description of why the *Service* failed. The additional parameters should include the details of the request.

The *Read* and *HistoryRead Services* may generate audit entries and audit *Events* for failed *Service* invocations. These *Services* should generate an audit *Event* of type *AuditEventType* or a subtype of it. See IEC 62541-5 for the detailed assignment of the *SourceNode*, *SourceName* and additional parameters. The *Message* for *Events* of this type should include a description of why the *Service* failed.

For *Clients* that support auditing, accessing the *Write* or *HistoryUpdate* services in the *Attribute Service Set* shall generate audit entries for both successful and failed invocations of the *Service*. Invocations of the other *Services* in this *Service Set* may generate audit entries. All audit entries should be setup prior to the actual *Service* invocation, allowing the invocation to contain the correct audit record id.

6.2.9 Auditing for Method Service Set

All *Services* in this *Service Set* for *Servers* that support auditing may generate audit entries and shall generate audit *Events* for both successful and failed service invocations if the invocation modifies the address space, writes a value or modifies the state of the system

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

(alarm acknowledge, batch sequencing or other system changes). These method calls shall generate an audit *Event* of type *AuditUpdateMethodEventType* or subtypes of it. Methods that do not modify the address space, write values or modify the state of the system may generate events. See IEC 62541-5 for the detailed assignment of the *SourceNode*, *SourceName* and additional parameters.

For *Clients* that support auditing, accessing the *Method Service Set* shall generate audit entries for both successful and failed invocations of the *Service*, if the invocation modifies the address space, writes a value or modifies the state of the system (alarm acknowledge, batch sequencing or other system changes). Invocations of the other *Methods* may generate audit entries. All audit entries should be setup prior to the actual *Service* invocation, allowing the invocation to contain the correct audit record id.

6.2.10 Auditing for View, Query, MonitoredItem and Subscription Service Set

All of the *Services* in these four *Service Sets* only provide the *Client* with information, with the exception of the *TransferSubscriptions Service* in the *Subscription Service Set*. In general, these services will not generate audit entries or audit Event *Messages*. The *TransferSubscriptions Service* shall generate an audit *Event* of type *AuditSessionEventType* or subtypes of it for both successful and failed *Service* invocations. See IEC 62541-5 for the detailed assignment of the *SourceNode*, the *SourceName* and additional parameters. For the failure case, the *Message* for Events of this type should include a description of why the service failed.

For *Clients* that support auditing, accessing the *TransferSubscriptions Service* in the *Subscription Service Set* shall generate audit entries for both successful and failed invocations of the *Service*. Invocations of the other *Services* in this *Service Set* do not require audit entries. All audit entries should be setup prior to the actual *Service* invocation, allowing the invocation to contain the correct audit record id.

6.3 Redundancy

6.3.1 Redundancy overview

Redundancy in OPC UA ensures that both Clients and Server can be redundant. OPC UA does not provide redundancy, it provides the data structures and services by which redundancy may be achieved in a standardized manner.

6.3.2 Server redundancy overview

6.3.2.1 General

Server redundancy comes in two modes, transparent and non-transparent. By definition, in transparent redundancy the failover of *Server* responsibilities from one *Server* to another is transparent to the *Client*: the *Client* does not care or even know that failover has occurred; the *Client* does not need to do anything at all to keep data flowing. In contrast, non-transparent failover requires some activity on the part of the *Client*.

The two areas where redundancy creates specific needs are in keeping the *Server* and *Client* information synchronised across *Servers*, and in controlling the failover of data flow from one *Server* to another.

6.3.2.2 Transparent redundancy

For transparent redundancy, all that OPC UA provides is the data structures to allow the *Client* to identify what *Servers* are available in the redundant set, what the service level of each *Server* is and which *Server* is currently supporting a specified *Session*. All OPC UA interactions within a given session shall be supported by one *Server* and the *Client* is able to identify which *Server* that is, allowing a complete audit trail for the data. It is the responsibility of the *Servers* to ensure that information is synchronised between the *Servers* and another

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Server will take over the *Session* and *Subscriptions* of the address from one *Server* to another upon*Client* in the case of failover. Failover may require a transport layer reconnect of the *Client* but the *Endpoint* URL of the *Server* shall not change.

Figure 24 shows a typical transparent redundancy setup.

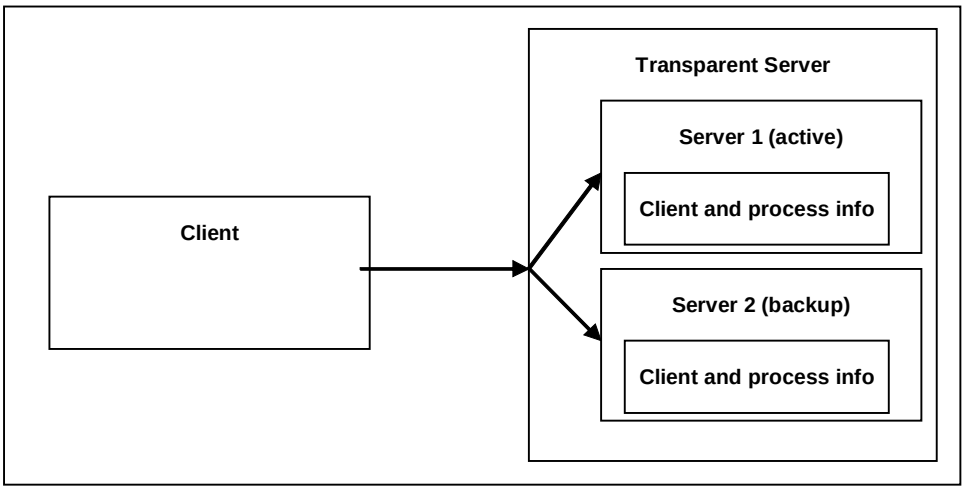


Figure 24 – Transparent Redundancy setup

6.3.2.3 Non-transparent redundancy

For non-transparent redundancy, OPC UA provides the same data structures and also *Server* information which tells the *Client* what modes of failover the *Server* supports. This information allows the *Client* to determine what actions it may need to take in order to accomplish failover.

Figure 25 shows a typical non-transparent redundancy setup.

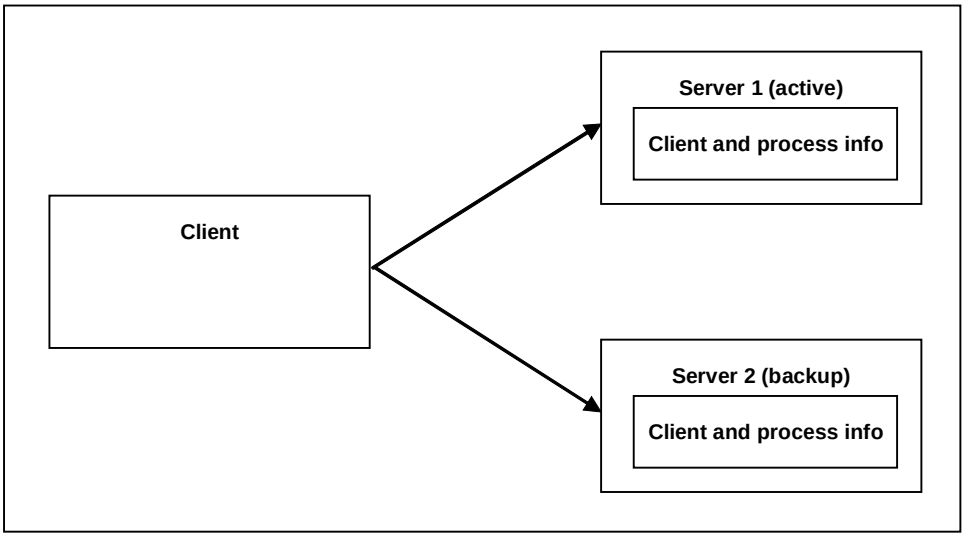


Figure 25 – Non-Transparent Redundancy setup

For non-transparent redundancy the *Server* has additional concepts of cold, warm and hot failover. Cold failovers are for *Servers* where only one *Server* can be active at a time. Warm failovers are for *Servers* where the backup *Servers* can be active, but cannot connect to actual data points (typically, a system where the underlying devices are limited to a single connection). Hot failovers are for *Servers* where more than one *Server* can be active and fully operational.

Table 102 defines the list of failover actions.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 102 – Redundancy failover actions

Failover mode	COLD	WARM	HOT
On initial connection:			
Connect to more than one OPC UA *Server*		X	X
Creating *Subscriptions* and adding monitored items to them		X	X
Activating sampling on the *Subscriptions*			X
At Failover:			
Connect to backup OPC UA *Server*	X		
Creating *Subscriptions* and adding monitored items	X		
Activating sampling on the *Subscriptions*	X	X	
Activate publishing	X	X	X

Some or all of that activity may be pushed into a *Server* proxy on the *Client* machine, to reduce the amount of functionality that shall be designed into the *Client* and to enable simpler *Clients* to take advantage of non-transparent redundancy. By using the *TransferSubscriptions Service*, which allows a *Client* to request that a set of *Subscriptions* be moved from one *Session* to another, a *Server* vendor can effectively make transparent failover a part of a proxy stub that lives on the *Client*. There are two ways to do this, one requiring code in the *Server* to support this and the other doing it all from the *Client* proxy process.

When the *Client* proxy is used, the proxy simply duplicates *Subscriptions* and modifications to *Subscriptions*, by passing the calls on to both *Servers*, but only enabling publishing or sampling on one *Server*. When the proxy detects a failure, it enables publishing and/or sampling on the backup *Server*, just as the *Client* would if it were a redundancy-aware *Client*.

The other method also requires a *Client* stub, but in this case the stub is a much lighter-weight process. In this mode, it is the *Server* which mirrors all *Subscriptions* in the other *Server*, but the *Client* endpoint for these *Subscriptions* is the active *Server*. When the stub detects that the active *Server* has failed, it issues a *TransferSubscriptions* call to the backup *Server*, moving the *Subscriptions* from the *Session* owned by the failed *Server* to its own *Session*, and activating publishing.

Figure 26 shows the difference between *Client* proxy and *Server* proxy redundancy.

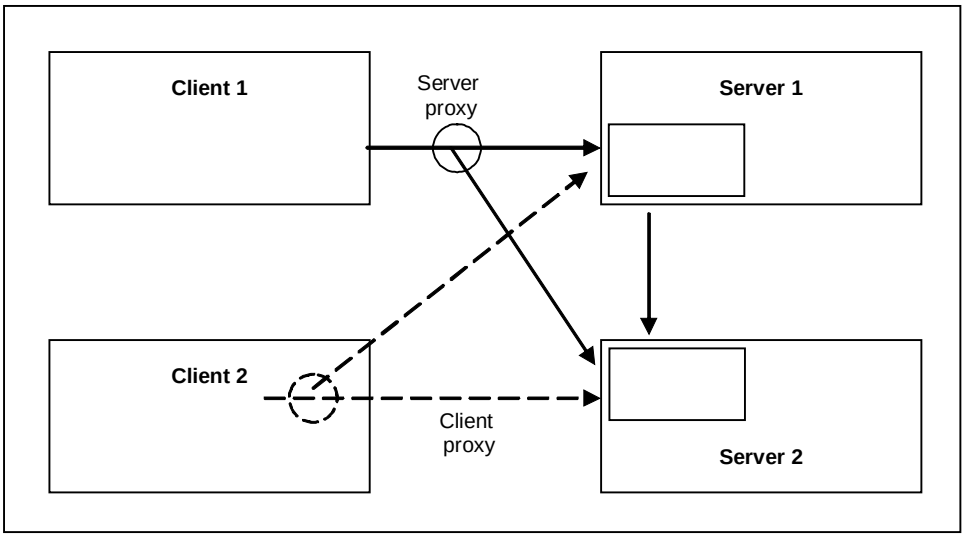


Figure 26 – Redundancy mode

6.3.3 Client redundancy

Client redundancy is supported in OPC UA by the *TransferSubscriptions* call and by exposing *Client* information in the *Server* information structures. Since *Subscription* lifetime is not tied to the *Session* in which it was created, backup *Clients* can monitor the active *Client's Session* with the *Server*, just as they would monitor any other data variable. If the active *Client* ceases

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

to be active, the *Server* shall send a data update to any *Client* which has that variable monitored. Upon receiving such notification, a backup *Client* would then instruct the *Server* to transfer the *Subscriptions* to its own session. If the *Subscription* is crafted carefully, with sufficient resources to buffer data during the change-over, there need be no data loss from a *Client* failover.

OPC UA does not provide a standardized mechanism for conveying the *SessionId* and *SubscriptionIds* from the active *Client* to the backup *Clients*, but as long as the backup *Clients* know the *Client* name of the active *Client*, this information is readily available using the *SessionDiagnostics* and *SubscriptionDiagnostics* portions of the *ServerDiagnostics* data.

7 Common parameter type definitions

7.1 ApplicationDescription

The components of this parameter are defined in Table 103.

Table 103 – ApplicationDescription

Name	Type	Description
ApplicationDescription	structure	Specifies an application that is available.
applicationUri	String	The globally unique identifier for the application instance.
productUri	String	The globally unique identifier for the product.
applicationName	LocalizedText	A localized descriptive name for the application.
applicationType	Enum ApplicationType	The type of application. This value is an enumeration with one of the following values: SERVER_0 The application is a *Server*. CLIENT_1 The application is a *Client*. CLIENTANDSERVER_2 The application is a *Client* and a *Server*. DISCOVERYSERVER_3 The application is a *DiscoveryServer.*
gatewayServerUri	String	A URI that identifies the *Gateway Server* associated with the *discoveryUrls*. This value is not specified if the *Server* can be accessed directly. This field is not used if the *applicationType* is CLIENT_1.
discoveryProfileUri	String	A URI that identifies the discovery profile supported by the URLs provided. This field is not used if the *applicationType* is CLIENT_1. If this value is not specified then the Endpoints shall support the Discovery Services defined in 5.4. Alternate discovery profiles are defined in IEC 62541-7.
discoveryUrls []	String	A list of URLs for the discovery *Endpoints* provided by the application. If the *applicationType* is CLIENT_1, this field shall contain an empty list.

7.2 ApplicationInstanceCertificate

An *ApplicationInstanceCertificate* is a *ByteString* containing an encoded *Certificate.* The encoding of an *ApplicationInstanceCertificate* depends on the security technology mapping and is defined completely in IEC 62541-6. Table 104 specifies the information that shall be contained in an *ApplicationInstanceCertificate*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 104 – ApplicationInstanceCertificate

Name	Type	Description
ApplicationInstanceCertificate	structure	*ApplicationInstanceCertificate* with signature created by a *Certificate Authority*.
version	String	An identifier for the version of the *Certificate* encoding.
serialNumber	ByteString	A unique identifier for the *Certificate* assigned by the Issuer.
signatureAlgorithm	String	The algorithm used to sign the *Certificate*. The syntax of this field depends on the *Certificate* encoding.
signature	ByteString	The signature created by the Issuer.
issuer	Structure	A name that identifies the Issuer *Certificate* used to create the signature.
validFrom	UtcTime	When the *Certificate* becomes valid.
validTo	UtcTime	When the *Certificate* expires.
subject	Structure	A name that identifies the application instance that the *Certificate* describes. This field shall contain the *productName* and the name of the organization responsible for the application instance.
applicationUri	String	The *applicationUri* specified in the *ApplicationDescription*. The *ApplicationDescription* is described in 7.1.
hostnames []	String	The name of the machine where the application instance runs. A machine may have multiple names if it is accessible via multiple networks. The hostname may be a numeric network address or a descriptive name. *Server Certificates* shall have at least one hostname defined.
publicKey	ByteString	The public key associated with the *Certificate*.
keyUsage []	String	Specifies how the *Certificate* key may be used. *ApplicationInstanceCertificates* shall support Digital Signature, Non-Repudiation Key Encryption, Data Encryption and Client/Server Authorization. The contents of this field depend on the *Certificate* encoding.

7.3 BrowseResult

The components of this parameter are defined in Table 105.

Table 105 – BrowseResult

Name	Type	Description
BrowseResult	structure	The results of a Browse operation.
statusCode	StatusCode	The status for the *BrowseDescription.* This value is set to *Good_MoreReferencesExist* if there are still references to return for the *BrowseDescription*.
continuationPoint	ContinuationPoint	A *Server* defined opaque value that identifies the continuation point. The *ContinuationPoint* type is defined in 7.6.
References []	ReferenceDescription	The set of references that meet the criteria specified in the *BrowseDescription*. Empty, if no *References* met the criteria. The Reference Description type is defined in 7.24.

7.4 ContentFilter

7.4.1 ContentFilter structure

The *ContentFilter* structure defines a collection of elements that make up filtering criteria. Each element in the collection describes an operator and an array of operands to be used by the operator. The operators that can be used in a ContentFilter are described in Table 110. The filter is evaluated by evaluating the first entry in the element array starting with the first operand in the operand array. The operands of an element may contain *References* to sub-elements resulting in the evaluation continuing to the referenced elements in the element array. If an element cannot be traced back to the starting element it is ignored. Extra operands for any operator shall result in an error. Annex B provides examples using the *ContentFilter* structure.

Table 106 defines the ContentFilter structure.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 106 – ContentFilter Structure

Name	Type	Description
ContentFilter	structure	
elements []	ContentFilterElement	List of operators and their operands that compose the filter criteria. The filter is evaluated by starting with the first entry in this array.
filterOperator	enum FilterOperator	Filter operator to be evaluated. The *FilterOperator* enumeration is defined in Table 110.
filterOperands []	Extensible Parameter FilterOperand	Operands used by the selected operator. The number and use depend on the operands defined in Table 110. This array needs at least one entry. This extensible parameter type is the *FilterOperand* parameter type specified in 7.4.4. It specifies the list of valid *FilterOperand* values.

7.4.2 ContentFilterResult

The components of this data type are defined in Table 107.

Table 107 – ContentFilterResult Structure

Name	Type	Description
ContentFilterResult	structure	A structure that contains any errors associated with the filter.
elementResults	ContentFilter ElementResult	A list of results for individual elements in the filter. The size and order of the list matches the size and order of the elements in the *ContentFilter* parameter.
statusCode	StatusCode	The status code for a single element.
operandStatusCodes []	StatusCode	A list of status codes for the operands in an element. The size and order of the list matches the size and order of the operands in the *ContentFilterElement.* This list is empty if no operand errors occurred.
operandDiagnosticInfos []	DiagnosticInfo	A list of diagnostic information for the operands in an element. The size and order of the list matches the size and order of the operands in the *ContentFilterElement.* 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 operands.
elementDiagnosticInfos []	DiagnosticInfo	A list of diagnostic information for individual elements in the filter. The size and order of the list matches the size and order of the elements in the *filter* 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 elements.

Table 108 defines values for the *statusCode* parameter that are specific to this structure. Common *StatusCodes* are defined in Table 166.

Table 108 – ContentFilterResult Result Codes

Symbolic Id	Description
Bad_FilterOperandCountMismatch	The number of operands provided for the filter operator was less than expected for the operand provided.
Bad_FilterOperatorInvalid	An unrecognized operator was provided in a filter.
Bad_FilterOperatorUnsupported	A valid operator was provided, but the server does not provide support for this filter operator.

Table 109 defines values for the operand*StatusCode* parameter that are specific to this structure. Common *StatusCodes* are defined in Table 166.

Table 109 – ContentFilterResult Operand Result Codes

Symbolic Id	Description
Bad_FilterOperandInvalid	See Table 166 for the description of this result code.
Bad_FilterElementInvalid	The referenced element is not a valid element in the content filter.
Bad_FilterLiteralInvalid	The referenced literal is not a valid basedatatype.
Bad_AttribteIdInvalid	The attribute id is not a valid attribute id in the system.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.
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_NotTypeDefinition	The provided Nodeid was not a type definition nodeid.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.4.3 FilterOperator

Table 110 defines the basic operators that can be used in a ContentFilter. See Table 111 for a description of advanced operators. See 7.4.4 for a definition of operands.

Table 110 – Basic FilterOperator Definition

Operator	Number of Operands	Description
Equals_0	2	TRUE if operand[0] is equal to operand[1]. If the operands are of different types, the system shall perform any implicit conversion to a common type. This operator resolves to FALSE if no implicit conversion is available and the operands are of different types. This operator returns FALSE if the implicit conversion fails. See the discussion on data type precedence in Table 114 for more information how to convert operands of different types.
IsNull_1	1	TRUE if operand[0] is a null value.
GreaterThan_2	2	TRUE if operand[0] is greater than operand[1]. The following restrictions apply to the operands: [0]: Any operand that resolves to an ordered value. [1]: Any operand that resolves to an ordered value. The same conversion rules as defined for *Equals* apply.
LessThan_3	2	TRUE if operand[0] is less than operand[1]. The same conversion rules and restrictions as defined for *GreaterThan* apply.
GreaterThanOrEqual_4	2	TRUE if operand[0] is greater than or equal to operand[1]. The same conversion rules and restrictions as defined for *GreaterThan* apply.
LessThanOrEqual_5	2	TRUE if operand[0] is less than or equal to operand[1]. The same conversion rules and restrictions as defined for *GreaterThan* apply.
Like_6	2	TRUE if operand[0] matches a pattern defined by operand[1]. See Table 112 for the definition of the pattern syntax. The following restructions apply to the operands: [0]: Any operand that resolves to a String. [1]: Any operand that resolves to a String. This operator resolves to FALSE if no operand can be resolved to a string.
Not_7	1	TRUE if operand[0] is FALSE. The following restrictions apply to the operands: [0]: Any operand that resolves to a Boolean. If the operand cannot be resolved to a Boolean, the result is a NULL. See below for a discussion on the handling of NULL.
Between_8	3	TRUE if operand[0] is greater or equal to operand[1] and less than or equal to operand[2]. The following restructions apply to the operands: [0]: Any operand that resolves to an ordered value. [1]: Any operand that resolves to an ordered value. [2]: Any operand that resolves to an ordered value. If the operands are of different types, the system shall perform any implicit conversion to match all operands to a common type. If no implicit conversion is available and the operands are of different types, the particular result is FALSE. See the discussion on data type precedence in Table 114 for more information how to convert operands of different types.
InList_9	2..n	TRUE if operand[0] is equal to one or more of the remaining operands. The Equals Operator is evaluated for operand[0] and each remaining operand in the list. If any Equals evaluation is TRUE, InList returns TRUE.
And_10	2	TRUE if operand[0] and operand[1] are TRUE. The following restructions apply to the operands: [0]: Any operand that resolves to a Boolean. [1]: Any operand that resolves to a Boolean. If any operand cannot be resolved to a Boolean it is considered NULL. See below for a discussion on the handling of NULL.
Or_11	2	TRUE if operand[0] or operand[1] are TRUE.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Operator	Number of Operands	Description
		The following restructions apply to the operands: [0]: Any operand that resolves to a Boolean. [1]: Any operand that resolves to a Boolean. If any operand cannot be resolved to a Boolean it is considered a NULL. See below for a discussion on the handling of NULL.
Cast_12	2	Converts operand[0] to a value with a data type with a NodeId identified by operand[1]. The following restructions apply to the operands: [0]: Any operand. [1]: Any operand that resolves to a NodeId or ExpandedNodeId where the *Node* is of the *NodeClass DataType*. If there is any error in conversion or in any of the parameters then the Cast Operation evaluates to a NULL. See below for a discussion on the handling of NULL.
BitwiseAnd_16	2	The result is an integer which matches the size of the largest operand and contains a bitwise And operation of the two operands where both have been converted to the lsame size (largest of the two operands). The following restructions apply to the operands: [0]: Any operand that resolves to a integer. [1]: Any operand that resolves to a integer. If any operand cannot be resolved to a integer it is considered a NULL. See below for a discussion on the handling of NULL.
BitwiseOr_17	2	The result is an integer which matches the size of the largest operand and contains a bitwise Or operation of the two operands where both have been converted to the lsame size (largest of the two operands). The following restructions apply to the operands: [0]: Any operand that resolves to a Integer. [1]: Any operand that resolves to a Integer. If any operand cannot be resolved to a Integer it is considered a NULL. See below for a discussion on the handling of NULL.

Many operands have restrictions on their type. This requires that the operand be evaluated to determine what type it is. In some cases, it is the type specified in the operand (i.e. a LiteralOperand). In other cases the type requires that the value of an attribute be read. ElementOperands are Boolean values unless the operator is Cast or a nested *RelatedTo* operator.

Table 111 defines complex operators that require a target node (i.e. row) to evaluate. These operators shall be re-evaluated for each possible target node in the result set.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 111 – Complex FilterOperator Definition

Operator	Number of Operands	Description
InView_13	1	TRUE if the target *Node* is contained in the *View* defined by operand[0]. The following restrictions apply to the operands: [0]: Any operand that resolves to a NodeId that identifies a View Node. If operand[0] does not resolve to a NodeId that identifies a View Node, this operation shall always be False.
OfType_14	1	TRUE if the target *Node* is of type operand[0] or of a subtype of operand[0]. The following restrictions apply to the operands: [0]: Any operand that resolves to a NodeId that identifies an ObjectType or VariableType Node. If operand[0] does not resolve to a NodeId that identifies an ObjectType or VariableType Node, this operation shall always be False.
RelatedTo_15	6	TRUE if the target *Node* is of type Operand[0] and is related to a *NodeId* of the type defined in Operand[1] by the *Reference* type defined in Operand[2]. Operand[0] or Operand[1] can also point to an element *Reference* where the referred to element is another RelatedTo operator. This allows chaining of relationships (e.g. A is related to B is related to C). In this case, the referred to element returns a list of *NodeIds* instead of TRUE or FALSE. In this case if any errors occur or any of the operands cannot be resolved to an appropriate value, the result of the chained relationship is an empty list of nodes. Operand[3] defines the number of hops the relationship should be followed. If Operand[3] is 1, then objects shall be directly related. If a hop is greater than 1, then a *NodeId* of the type described in Operand[1] is checked for at the depth specified by the hop. In this case, the type of the intermediate *Node* is undefined, and only the *Reference* type used to reach the end *Node* is defined. If the requested number of hops cannot be followed, then the result is FALSE, i.e., an empty *Node* list. If Operand[3] is 0, the relationship is followed to its logical end in a forward direction and each *Node* is checked to be of the type specified in Operand[1]. If any *Node* satisfies these criteria, then the result is TRUE, i.e., the *NodeId* is included in the sub-list. Operand[4] defines if Operands [0] and [1] should include support for subtypes of the types defined by these operands. A TRUE indicates support for subtypes. Operand[5] defines if Operand[3] should include support for subtypes of the reference type defined by Operand[3]. A TRUE indicates support for subtypes. The following restrictions apply to the operands: [0]: Any operand that resolves to a NodeId or ExpandedNodeId that identifies an ObjectType or VariableType Node or a reference to another element which is a RelatedTo operator. [1]: Any operand that resolves to a NodeId or ExpandedNodeId that identifies an ObjectType or VariableType Node or a reference to another element which is a RelatedTo operator. [2]: Any operand that resolves to a NodeId that identifies a ReferenceType Node. [3]: Any operand that resolves to a value implicitly convertible to Int32. [4]: Any operand that resolves to a value implicitly convertible to a boolean; if this operand does not resolve to a Boolean, then a value of FALSE is used. [5]: Any operand that resolves to a value implicitly convertible to a boolean; if this operand does not resolve to a Boolean, then a value of FALSE is used. If none of the operands [0],[1],[2],[3] resolves to an appropriate value then the result of this operation shall always be False (or an Empty set in the case of a nested relatedTo operand). See examples for RelatedTo in 7.4.4.

The RelatedTo operator can be used to identify if a given type, set as operand[1], is a subtype of another type set as operand[0] by setting operand[2] to the *HasSubtype ReferenceType* and operand[3] to 0.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The *Like* operator can be used to perform wildcard comparisons. Several special characters can be included in the second operand of the *Like* operator. The valid characters are defined in Table 112. The wildcard characters can be combined in a single string (i.e. 'Th[ia][ts]%' would match 'That is fine', 'This is fine', 'That as one', 'This it is', 'Then at any', etc.).

Table 112 – Wildcard characters

Special character	Description
%	Matches any string of zero or more characters (i.e. 'main%' would match any string that starts with 'main', '%en%' would match any string that contains the letters 'en' such as 'entail', 'green' and 'content'.) If a '%' sign is intended in a string the list operand can be used (i.e. 5[%] would match '5%').
_	Matches any single character (i.e. '_ould' would match 'would', 'could'). If the '_' is indended in a string then the list operand can be used (i.e. 5[_] would match '5_').
\	Escape character allows literal interpretation (i.e. \\ is \, \% is %, _ is _).
[]	Matches any single character in a list (i.e. 'abc[13-68] would match 'abc1','abc3','abc4','abc5','abc6', and 'abc8'. 'xyz[c-f]' would match 'xyzc', 'xyzd', 'xyze', 'xyzf').
[^]	Not Matching any single character in a list. The ^ shall be the first character inside the []. (i.e. 'ABC[^13-5]' would NOT match 'ABC1', 'ABC3', 'ABC4', and 'ABC5'. xyz[^dgh] would NOT match 'xyzd', 'xyzg', 'xyzh'.)

Table 113 defines the conversion rules for the operand values. The types are automatically converted if an implicit conversion exists (I). If an explicit conversion exists (E) then type can be converted with the cast operator. If no conversion exists (X) the then types cannot be converted, however, some servers may support application specific explicit conversions. The types used in the table are defined in IEC 62541-3. A data type that is not in the table does not have any defined conversions.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Table 113 – Conversion Rules

Source Type (From)	Target Type (To)																			
	Boolean	Byte	ByteString	DateTime	Double	ExpandedNodeId	Float	Guid	Int16	Int32	Int64	NodeId	SByte	StatusCode	String	LocalizedText	QualifiedName	UInt16	UInt32	UInt64	XmlElement
Boolean	-	I	X	X	I	X	I	X	I	I	I	X	I	X	E	X	X	I	I	I	X
Byte	E	-	X	X	I	X	I	X	I	I	I	X	I	X	E	X	X	I	I	I	X
ByteString	X	X	-	X	X	X	X	E	X	X	X	X	X	X	X	X	X	X	X	X	X
DateTime	X	X	X	-	X	X	X	X	X	X	X	X	X	X	E	X	X	X	X	X	X
Double	E	E	X	X	-	X	E	X	E	E	E	X	E	X	E	X	X	E	E	E	X
ExpandedNodeId	X	X	X	X	X	-	X	X	X	X	X	E	X	X	I	X	X	X	X	X	X
Float	E	E	X	X	I	X	-	X	E	E	E	X	E	X	E	X	X	E	E	E	X
Guid	X	X	E	X	X	X	X	-	X	X	X	X	X	X	E	X	X	X	X	X	X
Int16	E	E	X	X	I	X	I	X	-	I	I	X	E	X	E	X	X	E	I	I	X
Int32	E	E	X	X	I	X	I	X	E	-	I	X	E	E	E	X	X	E	E	I	X
Int64	E	E	X	X	I	X	I	X	E	E	-	X	E	E	E	X	X	E	E	E	X
NodeId	X	X	X	X	X	I	X	X	X	X	X	-	X	X	I	X	X	X	X	X	X
SByte	E	E	X	X	I	X	I	X	I	I	I	X	-	X	E	X	X	I	I	I	X
StatusCode	X	X	X	X	X	X	X	X	X	I	I	X	X	-	X	X	X	E	I	I	X
String	I	I	X	E	I	E	I	I	I	I	I	E	I	X	-	E	E	I	I	I	X
LocalizedText	X	X	X	X	X	X	X	X	X	X	X	X	X	X	I	-	X	X	X	X	X
QualifiedName	X	X	X	X	X	X	X	X	X	X	X	X	X	X	I	I	-	X	X	X	X
UInt16	E	E	X	X	I	X	I	X	I	I	I	X	E	I	E	X	X	-	I	I	X
UInt32	E	E	X	X	I	X	I	X	E	I	I	X	E	E	E	X	X	E	-	I	X
UInt64	E	E	X	X	I	X	I	X	E	E	I	X	E	E	E	X	X	E	E	-	X
XmlElement	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-

Arrays of a source type can be converted to arrays of the target type by converting each element. A conversion error for any element causes the entire conversion to fail.

Arrays of length 1 can be implicitly converted to a scalar value of the same type.

Guid, NodeIds and ExpandedNodeIds are converted to and from Strings, using the syntax defined in IEC 62541-6.

Floating point values are rounded by adding 0,5 and truncating when they are converted to integer values.

ByteStrings are converted to Strings by formatting the bytes as a sequence of hexadecimal digits.

LocalizedText values are converted to Strings by dropping the Locale. Strings are converted to LocalizedText values by setting the Locale to “”.

QualifiedName values are converted to Strings by dropping the NamespaceIndex. Strings are converted to QualifiedName values by setting the NamespaceIndex to 0.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

A StatusCode can be converted to and from a UInt32 and Int32 by copying the bits. Only the top 16-bits if the StatusCode are copied when it is converted to and from a UInt16 or Int16 value.

Boolean values are converted to '1' when true and '0' when false. Non zero numeric values are converted to true Boolean values. Numeric values of 0 are converted to false Boolean values. String values containing "true", "false", "1" or "0" can be converted to Boolean values. Other string values cause a conversion error. In this case strings are case-insensitive.

It is sometimes possible to use implicit casts when operands with different data types are used in an operation. In this situation the precedence rules defined in Table 114 are used to determine which implicit conversion to use. The first data type in the list (top down) has the most precedence. If a data type is not in this table then it cannot be converted implicitly while evaluating an operation.

For example, assume that A = 1.1 (Float) and B = 1 (Int32) and that these values are used with an Equals operator. This operation would be evaluated by casting the Int32 value to a Float since the Float data type has more precedence.

Table 114 – Data Precedence Rules

Rank	Data Type
1	Double
2	Float
3	Int64
4	UInt64
5	Int32
6	UInt32
7	StatusCode
8	Int16
9	UInt16
10	SByte
11	Byte
12	Boolean
13	Guid
14	String
15	ExpandedNodeId
16	NodeId
17	LocalizedText
18	QualifiedName

Operands may contain null values (i.e. values which do not exist). When this happens the element always evaluates to NULL (unless the IsNull operator has been specified). Table 115 defines how to combine elements that evaluate to NULL with other elements in a logical AND operation.

Table 115 – Logical AND Truth Table

	TRUE	**FALSE**	**NULL**
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Table 116 defines how to combine elements that evaluate to NULL with other elements in a logical OR operation.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 116 – Logical OR Truth Table

	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

The NOT operator always evaluates to NULL if applied to a NULL operand.

A ContentFilter which evaluates to NULL after all elements are evaluated is evaluated as false.

7.4.4 FilterOperand parameters

7.4.4.1 Overview

The *ContentFilter* structure specified in 7.4 defines a collection of elements that makes up filter criteria and contains different types of *FilterOperands*. The *FilterOperand* parameter is an extensible parameter. This parameter is defined in Table 117. The *ExtensibleParamter* type is defined in 7.11.

Table 117 – FilterOperand parameterTypeIds

Symbolic Id	Description
Element	Specifies an index into the array of elements. This type is used to build a logic tree of sub-elements by linking the operand of one element to a sub-element.
Literal	Specifies a literal value.
Attribute	Specifies any *Attribute* of an *Object* or *Variable Node* using a *Node* in the type system and relative path constructed from *ReferenceTypes* and *BrowseNames*.
SimpleAttribute	Specifies any *Attribute* of an *Object* or *Variable Node* using a *TypeDefinition* and a relative path constructed from *BrowseNames*.

7.4.4.2 ElementOperand

The *ElementOperand* provides the linking to sub-elements within a *ContentFilter*. The link is in the form of an integer that is used to index into the array of elements contained in the *ContentFilter*. An index is considered valid if its value is greater than the element index it is part of and it does not *Reference* a non-existent element. *Clients* shall construct filters in this way to avoid circular and invalid *References*. *Servers* should protect against invalid indexes by verifying the index prior to using it.

Table 118 defines the *ElementOperand* type.

Table 118 – ElementOperand

Name	Type	Description
ElementOperand	structure	ElementOperand value.
index	UInt32	Index into the element array.

7.4.4.3 LiteralOperand

Table 119 defines the *LiteralOperand* type.

Table 119 – LiteralOperand

Name	Type	Description
LiteralOperand	structure	LiteralOperand value.
value	BaseDataType	A literal value.

7.4.4.4 AttributeOperand

Table 120 defines the *AttributeOperand* type.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 120 – AttributeOperand

Name	Type	Description
AttributeOperand	structure	Attribute of a *Node* in the address space.
nodeId	NodeId	*NodeId* of a *Node* from the type system.
alias	String	An optional parameter used to identify or refer to an alias. An alias is a symbolic name that can be used to alias this operand and use it in other location in the filter structure.
browsePath	RelativePath	Browse path relative to the *Node* identified by the *nodeId* parameter. See 7.25 for the definition of *RelativePath*.
attributeId	IntegerId	Id of the *Attribute*. This shall be a valid *Attribute* id. The *IntegerId* is defined in 7.13. The IntegerIds for the Attributes are defined in IEC 62541-6.
indexRange	NumericRange	This parameter is used to identify a single element of an array or a single range of indexes for an array. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is not used if the specified *Attribute* is not an array. However, if the specified *Attribute* is an array and this parameter is not used, then all elements are to be included in the range. The parameter is null if not used.

7.4.4.5 SimpleAttributeOperand

The *SimpleAttributeOperand* is a simplified form of the *AttributeOperand* and all of the rules that apply to the *AttributeOperand* also apply to the *SimpleAttributeOperand*. The examples provided in Clause B.1 only use *AttributeOperands*, however, the *AttributeOperand* can be replaced by a *SimpleAttributeOperand* whenever all *ReferenceTypes* in the *RelativePath* are subtypes of *HierarchialReferences* and the targets are *Object* or *Variable Nodes and an Alias is not required*.

Table 121 defines the *SimpleAttributeOperand* type.

Table 121 – SimpleAttributeOperand

Name	Type	Description
SimpleAttributeOperand	structure	Attribute of a *Node* in the address space.
typeId	NodeId	*NodeId* of a *TypeDefinitionNode*. This parameter restricts the operand to instances of the *TypeDefinitionNode* or one of its subtypes.
browsePath []	QualifiedName	A relative path to a *Node*. This parameter specifies a relative path using a list of *BrowseNames* instead of the *RelativePath* structure used in the *AttributeOperand*. The list of *BrowseNames* is equivalent to a *RelativePath* that specifies forward references which are subtypes of the *HierarchicalReferences ReferenceType*. All *Nodes* followed by the *browsePath* shall be of the *NodeClass Object* or *Variable*. If this list is empty the *Node* is the instance of the *TypeDefinition*.
attributeId	IntegerId	Id of the *Attribute*. The *IntegerId* is defined in 7.13. The *Value Attribute* shall be supported by all *Servers*. The support of other *Attributes* depends on requirements set in Profiles or other parts of this specification.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for an array. The first element is identified by index 0 (zero). This parameter is ignored if the selected Node is not a Variable or the Value of a Variable is not an array. The parameter is null if not specifed. All values in the array are used if this parameter is not specified. The *NumericRange* type is defined in 7.21.

7.5 Counter

This primitive data type is a UInt32 that represents the value of a counter. The initial value of a counter is specified by its use. Modulus arithmetic is used for all calculations, where the modulus is max value + 1. Therefore,

$x + y = (x + y) \text{mod}(\text{max value} + 1)$

For example:

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

max value + 1 = 0

max value + 2 = 1

7.6 ContinuationPoint

A *ContinuationPoint* is used to pause a *Browse or QueryFirst* operation and allow it to be restarted later by calling *BrowseNext* or *QueryNext.* Operations are paused when the number of results found exceeds the limits set by either the *Client* or the *Server*.

The *Client* specifies the maximum number of results per operation in the request message. A *Server* shall not return more than this number of results but it may return fewer results. The *Server* allocates a *ContinuationPoint* if there are more results to return. The *Server* shall always return at least one result if it returns a *ContinuationPoint*.

Servers shall support at least one *ContinuationPoint* per *Session*. *Servers* specify a maximum number of *ContinuationPoints* per *Session* in the *ServerCapabilities Object* defined in IEC 62541-5. *ContinuationPoints* remain active until the *Client* retrieves the remaining results, the *Client* releases the *ContinuationPoint* or the *Session* is closed. A *Server* shall automatically free *ContinuationPoints* from prior requests if they are needed to process a new request. The *Server* returns a *Bad_ContinuationPointInvalid* error if a *Client* tries to use a *ContinuationPoint* that has been released.

Requests will often specify multiple operations that may or may not require a *ContinuationPoint*. A *Server* shall process the operations until it runs out of *ContinuationPoints*. Once that happens the *Server* shall return a *Bad_NoContinuationPoints* error for any remaining operations.

A *Client* restarts an operation by passing the *ContinuationPoint* back to the *Server*. The *Server* should always be able to reuse the *ContinuationPoint* provided, and so *Servers* shall never return *Bad_NoContinuationPoints* error when continuing a previously halted operation.

A *ContinuationPoint* is a subtype of the *ByteString* data type.

7.7 DataValue

7.7.1 General

The components of this parameter are defined in Table 122.

Table 122 – DataValue

Name	Type	Description
DataValue	structure	The value and associated information.
value	BaseDataType	The data value.
statusCode	StatusCode	The *StatusCode* that defines with the *Server*'s ability to access/provide the value. The *StatusCode* type is defined in 7.33
sourceTimestamp	UtcTime	The source timestamp for the value.
sourcePicoSeconds	UInteger	Specifies the number of 10 ps ($1{,}0 \times 10^{-11}$ s) intervals which shall be added to the sourceTimestamp.
serverTimestamp	UtcTime	The *Server* timestamp for the value.
serverPicoSeconds	UInteger	Specifies the number of 10 ps ($1{,}0 \times 10^{-11}$ s) intervals which shall be added to the serverTimestamp.

7.7.2 PicoSeconds

Some applications require high resolution timestamps. The *PicoSeconds* fields allow applications to specify timestamps with 10 ps resolution. The actual size of the *PicoSeconds* field depends on the resolution of the *UtcTime DataType*. For example, if the *UtcTime DataType* has a resolution of 100 ns then the *PicoSeconds* field would have to store values

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

,up to 10 000 in order to provide the 10 ps resolution. The resolution of the *UtcTime DataType* depends on the *Mappings* defined in IEC 62541-6.

7.7.3 SourceTimestamp

The *sourceTimestamp* is used to reflect the timestamp that was applied to a *Variable* value by the data source. Once a value has been assigned a source timestamp, the source timestamp for that value instance never changes. In this context, “value instance” refers to the value received, independent of its actual value.

The *sourceTimestamp* shall be UTC time and should indicate the time of the last change of the *value* or *statusCode*.

The *sourceTimestamp* should be generated as close as possible to the source of the value but the timestamp needs to be set always by the same physical clock. In the case of redundant sources, the clocks of the sources should be synchronised.

If the OPC UA *Server* receives the *Variable* value from another OPC UA *Server*, then the OPC UA *Server* shall always pass the source timestamp without changes. If the source that applies the timestamp is not available, the source timestamp is set to null. For example, if a value could not be read because of some error during processing like invalid arguments passed in the request, then the sourceTimestamp shall be null.

In the case of a bad or uncertain status *sourceTimestamp* is used to reflect the time that the source recognized the non-good status or the time the *Server* last tried to recover from the bad or uncertain status.

The *sourceTimestamp* is only returned with a *Value Attribute*. For all other *Attributes* the returned *sourceTimestamp* is set to null.

7.7.4 ServerTimestamp

The *serverTimestamp* is used to reflect the time that the *Server* received a *Variable* value or knew it to be accurate.

In the case of a bad or uncertain status, *serverTimestamp* is used to reflect the time that the *Server* received the status or that the *Server* last tried to recover from the bad or uncertain status.

In the case where the OPC UA *Server* subscribes to a value from another OPC UA *Server*, each *Server* applies its own *serverTimestamp*. This is in contrast to the *sourceTimestamp* in which only the originator of the data is allowed to apply the *sourceTimestamp*.

If the *Server* is subscribing to the value from another *Server* every ten seconds and the value changes, then the *serverTimestamp* is updated each time a new value is received. If the value does not change, then new values will not be received on the *Subscription*. However, in the absence of errors, the receiving *Server* applies a new *serverTimestamp* every ten seconds because not receiving a value means that the value has not changed. Thus, the *serverTimestamp* reflects the time at which the *Server* knew the value to be accurate.

This concept also applies to OPC UA *Servers* that receive values from exception-based data sources. For example, suppose that a *Server* is receiving values from an exception-based device, and that

a) the device is checking values every 0,5 s,

b) the connection to the device is good,

c) the device sent an update 3 min ago with a value of 1,234.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

In this case, the *Server* value would be 1,234 and the *serverTimestamp* would be updated every 0,5 s after the receipt of the value.

7.7.5 *StatusCode* assigned to a value

The *StatusCode* is used to indicate the conditions under which a *Variable* value was generated, and thereby can be used as an indicator of the usability of the value. The *StatusCode* is defined in 7.33.

Overall condition (severity)

- A *StatusCode* with severity Good means that the value is of good quality.
- A *StatusCode* with severity Uncertain means that the quality of the value is uncertain for reasons indicated by the Substatus.
- A *StatusCode* with severity Bad means that the value is not usable for reasons indicated by the Substatus.

Rules

- The *StatusCode* indicates the usability of the value. Therefore, It is required that *Clients* minimally check the *StatusCode Severity* of all results, even if they do not check the other fields, before accessing and using the value.
- A *Server*, which does not support status information, shall return a severity code of Good. It is also acceptable for a *Server* to simply return a severity and a non-specific (0) Substatus.
- If the *Server* has no known value - in particular when *Severity* is BAD, it shall return a NULL value.

7.8 DiagnosticInfo

The components of this parameter are defined in Table 123.

Table 123 – DiagnosticInfo

Name	Type	Description
DiagnosticInfo	structure	Vendor-specific diagnostic information.
identifier	structure	The vendor-specific identifier of an error or condition.
namespaceUri	Int32	The symbolic id defined by the *symbolicIdIndex* parameter is defined within the context of a namespace. This namespace is represented as a string and is conveyed to the *Client* in the *stringTable* parameter of the *ResponseHeader* parameter defined in 7.27. The *namespaceIndex* parameter contains the index into the *stringTable* for this string. -1 indicates that no string is specified.
symbolicId	Int32	A vendor-specific symbolic identifier string identifies an error or condition. The maximum length of this string is 32 characters. *Servers* wishing to return a numeric return code should convert the return code into a string and return the string in this identifier. This symbolic identifier string is conveyed to the *Client* in the *stringTable* parameter of the *ResponseHeader* parameter defined in 7.27. The *symbolicIdIndex* parameter contains the index into the *stringTable* for this string. -1 indicates that no string is specified.
locale	Int32	The locale part of the vendor-specific localized text describing the symbolic id. This localized text string is conveyed to the *Client* in the *stringTable* parameter of the *ResponseHeader* parameter defined in 7.27. The *localizedTextIndex* parameter contains the index into the *stringTable* for this string. -1 indicates that no string is specified.
localizedText	Int32	A vendor-specific localized text string describes the symbolic id. The maximum length of this text string is 256 characters. This localized text string is conveyed to the *Client* in the *stringTable* parameter of the *ResponseHeader* parameter defined in 7.27. The *localizedTextIndex* parameter contains the index into the *stringTable* for this string. -1 indicates that no string is specified.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

additionalInfo	String	Vendor-specific diagnostic information.
innerStatusCode	StatusCode	The *StatusCode* from the inner operation. Many applications will make calls into underlying systems during OPC UA request processing. An OPC UA *Server* has the option of reporting the status from the underlying system in the diagnostic info.
innerDiagnosticInfo	DiagnosticInfo	The diagnostic info associated with the inner *StatusCode*.

7.9 EndpointDescription

The components of this parameter are defined in Table 124.

Table 124 – EndpointDescription

Name	Type	Description
EndpointDescription	structure	Describes an *Endpoint* for a *Server*.
endpointUrl	String	The URL for the *Endpoint* described.
server	ApplicationDescription	The description for the *Server* that the *Endpoint* belongs to. The *ApplicationDescription* type is defined in 7.1.
serverCertificate	ApplicationInstance Certificate	The application instance *Certificate* issued to the *Server*. The *ApplicationInstanceCertificate* type is defined in 7.2.
securityMode	Enum MessageSecurityMode	The type of security to apply to the messages. The type *MessageSecurityMode* type is defined in 7.14. A *SecureChannel* may have to be created even if the *securityMode* is NONE. The exact behaviour depends on the mapping used and is described in the IEC 62541-6.
securityPolicyUri	String	The URI for *SecurityPolicy* to use when securing messages. The set of known URIs and the S*ecurityPolicies* associated with them are defined in IEC 62541-7.
userIdentityTokens []	UserTokenPolicy	The user identity tokens that the *Server* will accept. The *Client* shall pass one of the *UserIdentityTokens* in the *ActivateSession* request. The *UserTokenPolicy* type is described in 7.36.
transportProfileUri	String	The URI of the *Transport Profile* supported by the *Endpoint*. IEC 62541-7 defines URIs for the *Transport Profiles*.
securityLevel	Byte	A numeric value that indicates how secure the EndpointDescription is compared to other EndpointDescriptions for the same Server. A value of 0 indicates that the EndpointDescription is not recommended and is only supported for backward compatibility.

7.10 ExpandedNodeId

The components of this parameter are defined in Table 125. *ExpandedNodeIds* allow the namespace to be specified explicitly as a string or with an index in the *Server*'s namespace table.

Table 125 – ExpandedNodeId

Name	Type	Description
ExpandedNodeId	structure	The *NodeId* with the namespace expanded to its string representation.
serverIndex	Index	Index that identifies the *Server* that contains the *TargetNode*. This *Server* may be the local *Server* or a remote *Server*. This index is the index of that *Server* in the local *Server*'s *Server* table. The index of the local *Server* in the *Server* table is always 0. All remote *Servers* have indexes greater than 0. The *Server* table is contained in the *Server Object* in the *AddressSpace* (see IEC 62541-3 and IEC 62541-5). The *Client* may read the *Server* table *Variable* to access the description of the target *Server.*
namespaceUri	String	The URI of the namespace. If this parameter is specified then the namespace index is ignored. Subclause 5.4 and IEC 62541-12 describe discovery mechanism that can be used to resolve URIs into URLs.
namespaceIndex	Index	The index in the *Server*'s namespace table. This parameter shall be 0 and is ignored in the *Server* if the namespace URI is specified.
identifierType	IdType	Type of the identifier element of the *NodeId*.
identifier	*	The identifier for a *Node* in the address space of an OPC UA *Server* (see *NodeId* definition in IEC 62541-3).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.11 ExtensibleParameter

The extensible parameter types can only be extended by additional parts of this series of standards.

The *ExtensibleParameter* defines a data structure with two elements. The *parameterTypeId* specifies the data type encoding of the second element. Therefore the second element is specified as "--". The *ExtensibleParameter* base type is defined in Table 126.

Concrete extensible parameters that are common to OPC UA are defined in Clause 7. Additional parts of this series of standards can define additional extensible parameter types.

Table 126 – ExtensibleParameter Base Type

Name	Type	Description
ExtensibleParameter	structure	Specifies the details of an extensible parameter type.
parameterTypeId	NodeId	Identifies the data type of the parameter that follows.
parameterData	--	The details for the extensible parameter type.

7.12 Index

This primitive data type is an UInt32 that identifies an element of an array.

7.13 IntegerId

This primitive data type is an UInt32 that is used as an identifier, such as a handle. All values, except for 0, are valid.

7.14 MessageSecurityMode

The *MessageSecurityMode* is an enumeration that specifies what security should be applied to message exchanges during a Session. The possible values are described in Table 127.

Table 127 – MessageSecurityMode Values

Value	Description
INVALID_0	The MessageSecurityMode is invalid. This value is the default value to avoid an accidental choice of no security is applied. This choice will always be rejected.
SIGN_1	All messages are signed but not encrypted.
SIGNANDENCRYPT_2	All messages are signed and encrypted.

7.15 MonitoringParameters

The components of this parameter are defined in Table 128.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 128 – MonitoringParameters

Name	Type	Description
MonitoringParameters	structure	Parameters that define the monitoring characteristics of a *MonitoredItem*.
clientHandle	IntegerId	*Client*-supplied id of the *MonitoredItem*. This id is used in *Notifications* generated for the list *Node*. The *IntegerId* type is defined in 7.13.
samplingInterval	Duration	The interval that defines the fastest rate at which the *MonitoredItem*(s) should be accessed and evaluated. This interval is defined in milliseconds. The value 0 indicates that the *Server* should use the fastest practical rate. The value -1 indicates that the default sampling interval defined by the publishing rate of the *Subscription* is used. The *Server* uses this parameter to assign the *MonitoredItems* to a sampling interval that it supports.
filter	Extensible Parameter MonitoringFilter	A filter used by the *Server* to determine if the *MonitoredItem* should generate a *Notification*. If not used, this parameter is null. The *MonitoringFilter* parameter type is an extensible parameter type specified in 7.16. It specifies the types of filters that can be used.
queueSize	Counter	The requested size of the *MonitoredItem* queue. If *Events* are lost an *Event* of the type *EventQueueOverflow* is generated. The following values have special meaning: Value – Meaning 1 – the queue has a single entry, effectively disabling queuing. >1 – a first-in-first-out queue is to be used. Max Value – the max size that the *Server* can support. This is used for *Event Notifications*. In this case the *Server* is responsible for the *Event* buffer. If 0 is passed by the client, the server returns the default queue size to 1, and the return shall be 1 as *revisedQueueSize* for data monitored items. For event monitored items the value is ignored and the server shall set the supported maximum queue size as *revisedQueueSize*. If *Events* are lost an *Event* of the type *EventQueueOverflowEventType* is generated.
discardOldest	Boolean	A boolean parameter that specifies the discard policy when the queue is full and a new *Notification* is to be enqueued. It has the following values: TRUE – the oldest (first) *Notification* in the queue is discarded. The new *Notification* is added to the end of the queue. FALSE – the new *Notification* is discarded. The queue is unchanged.

7.16 MonitoringFilter parameters

7.16.1 Overview

The *CreateMonitoredItem Service* allows specifying a filter for each *MonitoredItem*. The *MonitoringFilter* is an extensible parameter whose structure depends on the type of item being monitored. The *parameterTypeIds* are defined in Table 129. Other types can be defined by additional parts of this multi-part specification or other specifications based on OPC UA. The *ExtensibleParamter* type is defined in 7.11.

Each *MonitoringFilter* may have an associated *MonitoringFilterResult* structure which returns revised parameters and/or error information to clients in the response. The result structures, when they exist, are described in the section that defines the *MonitoringFilter*.

Table 129 – MonitoringFilter parameterTypeIds

Symbolic Id	Description
DataChangeFilter	The change in a data value that shall cause a *Notification* to be generated.
EventFilter	If a *Notification* conforms to the *EventFilter*, the *Notification* is sent to the *Client*.
AggregateFilter	The aggregate and its intervals when it will be calculated and a Notification is generated.

7.16.2 DataChangeFilter

The *DataChangeFilter* defines the conditions under which a data change notification should be reported and, optionally, a range or band for value changes where no *DataChange Notification* is generated. This range is called *Deadband*. The *DataChangeFilter* is defined in Table 130.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 130 – DataChangeFilter

Name	Type	Description
DataChangeFilter	structure	
trigger	enum DataChangeTrigger	Specifies the conditions under which a data change notification should be reported. It has the following values: STATUS_0 Report a notification ONLY if the *StatusCode* associated with the value changes. See Table 166 for *StatusCodes* defined in this standard. IEC 62541-8 specifies additional *StatusCodes* that are valid in particular for device data. STATUS_VALUE_1 Report a notification if either the *StatusCode* or the value change. The *Deadband* filter can be used in addition for filtering value changes. **This is the default setting if no filter is set.** STATUS_VALUE_TIMESTAMP_2 Report a notification if either StatusCode, value or the *SourceTimestamp* change. The *Deadband* filter can be used in addition for filtering value changes. If the DataChangeFilter is not applied to the monitored item, STATUS_VALUE_1 is the default reporting behaviour.
deadbandType	UInt32	A value that defines the *Deadband* type and behaviour. Value deadbandType None_0 No *Deadband* calculation should be applied. Absolute_1 AbsoluteDeadband (see below) Percent_2 PercentDeadband (This type is specified in IEC 62541-8).
deadbandValue	Double	The *Deadband* is applied only if * the *trigger* includes value changes and * the *deadbandType* is set appropriately. Deadband is ignored if the status of the data item changes. ***DeadbandType = AbsoluteDeadband***: For this type the *deadbandValue* contains the absolute change in a data value that shall cause a *Notification* to be generated. This parameter applies only to *Variable*s with any number data type. An exception that causes a *DataChange Notification* based on an AbsoluteDeadband is determined as follows: Exception **if (absolute value of (last cached value - current value) > *AbsoluteDeadband*)** The last cached value is defined as the most recent value previously sent to the *Notification* channel. If the item is an array of values, the entire array is returned if any array element exceeds the AbsoluteDeadband. ***DeadbandType = PercentDeadband:*** This type is specified in IEC 62541-8.

The *DataChangeFilter does* not have an associated result structure.

7.16.3 EventFilter

The *EventFilter* provides for the filtering and content selection of *Event Subscriptions*.

If an *Event Notification* conforms to the filter defined by the *where* parameter of the *EventFilter*, then the *Notification* is sent to the *Client*.

Each *Event Notification* shall include the fields defined by the *selectClauses* parameter of the *EventFilter*. The defined *EventType*s are specified in IEC 62541-5.

The *selectClause* and *whereClause* parameters are specified with the *SimpleAttributeOperand* structure (see 7.4.4.5). This structure requires the *NodeId* of an *EventType* supported by the *Server* and a path to an *InstanceDeclaration*. An *InstanceDeclaration* is a *Node* which can be found by following forward hierarchical references from the fully inherited *EventType* where the *Node* is also the source of a *HasModellingRule* reference. *EventTypes*, *InstanceDeclarations* and *Modelling Rules* are described completely in IEC 62541-3.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

In some cases the same *BrowsePath* will apply to multiple *EventTypes*. If the *Client* specifies the *BaseEventType* in the SimpleAttributeOperand then the *Server* shall evaluate the *BrowsePath* without considering the *Type*.

Each *InstanceDeclaration* in the path shall be *Object* or *Variable Node*. The final *Node* in the path may be an *Object Node*, however, *Object Nodes* are only available for *Events* which are visible in the *Server's Address Space*.

The *SimpleAttributeOperand* structure allows the Client to specify any *Attribute*, however, the *Server* is only required to support the *Value Attribute* for *Variable Nodes* and the *NodeId Attribute* for *Object Nodes*. That said, profiles defined in IEC 62541-7 may make support for additional *Attributes* mandatory.

The *SimpleAttributeOperand* structure is used in the *selectClause* to select the value to return if an *Event* meets the criteria specified by the *whereClause*. A null value is returned in the corresponding event field in the Publish response if the selected *field* is not part of the *Event* or an error was returned in the *selectClauseResults* of the *EventFilterResult*. If the selected *field* is available but cannot be returned to *Client* then the *Server* shall return a *StatusCode* that indicates the reason for the error. For example, the *Server* shall return a *Bad_UserAccessDenied* error if the value is not accessible to the user associated with the *Session*. If a *Value Attribute* has an uncertain or bad *StatusCode* associated with it then the *Server* shall return the *StatusCode* instead of the *Value*.

The *Server* shall validate the *selectClauses* when a *Client* creates or updates the *EventFilter*. Any errors which are true for all possible *Events* are returned in the *selectClauseResults* parameter described in Table 132. The *Server* shall not report errors that might occur depending on the state or the *Server* or type of *Event*. For example, a *selectClause* that requests a single element in an array would always produce an error if the *DataType* of the *Attribute* is a scalar. However, even if the *DataType* is an array an error could occur if the requested index does not exist for a particular *Event*, the *Server* would not report an error in the *selectClauseResults* parameter if the latter situation existed.

The *SimpleAttributeOperand* is used in the *whereClause* to select a value which forms part of a logical expression. These logical expressions are then used to determine whether a particular *Event* should be reported to the *Client*. The *Server* shall use a null value if any error occurs when a *whereClause* is evaluated for a particular *Event*. If a *Value Attribute* has an uncertain or bad *StatusCode* associated with it then the *Server* shall use a null value instead of the *Value*.

Any basic *FilterOperator* in Table 110 may be used in the *whereClause*, however, only the OfType *FilterOperator* from Table 111 is permitted.

The *Server* shall validiate the *whereClauses* when a *Client* creates or updates the *EventFilter*. Any structural errors in the construction of the filter and any errors which are true for all possible *Events* are returned in the *whereClauseResult* parameter described in Table 132. Errors that could occur depending on the state of the *Server* or the *Event* are not reported.

SubscriptionControlEvents are special *Events* which are used to provide control information to the *Client*. These *Events* are only published to the *MonitoredItems* in the *Subscription* that produced the *SubscriptionControlEvent*. These *Events* bypass the *whereClause*.

Table 131 defines the EventFilter structure.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 131 – EventFilter structure

Name	Type	Description
EventFilter	structure	
selectClauses []	SimpleAttributeOperand	List of the values to return with each *Event* in a *Notification*. At least one valid clause shall be specified. See 7.4.4.5 for the definition of *SimpleAttributeOperand*.
whereClause	ContentFilter	Limit the *Notifications* to those *Events* that match the criteria defined by this ContentFilter. The ContentFilter structure is described in 7.4. The *AttributeOperand* structure may not be used in *EventFilters*.

Table 132 defines the EventFilterResult structure.

Table 132 – EventFilterResult structure

Name	Type	Description
EventFilterResult	structure	
selectClauseResults []	StatusCode	List of status codes for the elements in the select clause. The size and order of the list matches the size and order of the elements in the *selectClauses* request parameter. The Server returns null for unavailable or rejected *Event* fields.
selectClauseDiagnosticInfos []	DiagnosticInfo	A list of diagnostic information for individual elements in the select clause. The size and order of the list matches the size and order of the elements in the *selectClauses* 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 select clauses.
whereClauseResult	ContentFilterResult	A result associated with the *whereClause* request parameter. The *ContentFilterResult* type is defined in 7.4.2.

Table 133 defines values for the selectClauseResults parameter. Common *StatusCodes* are defined in Table 166.

Table 133 – EventFilterResult Result Codes

Symbolic Id	Description
Bad_TypeDefinitionInvalid	See Table 166 for the description of this result code. The typeId is not the NodeId for BaseEventType or a subtype of it.
Bad_NodeIdUnknown	See Table 166 for the description of this result code. The browsePath is specified but it will never exist in any Event.
Bad_BrowseNameInvalid	See Table 166 for the description of this result code. The browsePath is specified and contains a null element.
Bad_AttributeIdInvalid	See Table 166 for the description of this result code. The node specified by the browse path will never allow the given attribute id to be returned.
Bad_IndexRangeInvalid	See Table 166 for the description of this result code.
Bad_TypeMismatch	See Table 166 for the description of this result code. The indexRange is valid but the value of the Attribute is never an array.

7.16.4 AggregateFilter

The *AggregateFilter* defines the aggregate function that should be used to calculate the values to be returned. See IEC 62541-13 for details on possible aggregate functions. It specifies a startTime of the first aggregate to be calculated. The samplingInterval of the MonitoringAttributes (see 7.15) defines how the server should internally sample the underlying data source. The processingInterval specifies the size of a period where the aggregate is calculated. The queueSize from the MonitoringAttributes specifies the number of processed values that should be kept.

The intention of the AggregateFilter is not to read historical data, the HistoryRead service should be used for this purpose. However, it is allowed that the startTime is set to a time that is in the past when received from the server. The number of aggregates to be calculated in the past should not exceed the queueSize defined in the MonitoringAttributes since the values exceeding the queueSize would directly be discharged and never returned to the client.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The startTime and the processingInterval can be revised by the server, but the startTime should remain in the same boundary (startTime + revisedProcessingInterval * n = revisedStartTime). That behaviour simplifies accessing historical values of the aggregates using the same boundaries by calling the HistoryRead service. The extensible Parameter AggregateFilterResult is used to return the revised values for the AggregateFilter.

Some underlying systems may poll data and produce multiple samples with the same value. Other systems may only report changes to the values. The definition for each aggregate type explains how to handle the two different scenarios.

The *MonitoredItem* only reports values for intervals that have completed when the publish timer expires. Unused data is carried over and used to calculate a value returned in the next publish.

The *ServerTimestamp* for each interval shall be the time of the end of the processing interval.

The AggregateFilter is defined in Table 134.

Table 134 – AggregateFilter structure

Name	Type	Description
AggregateFilter	structure	
startTime	UtcTime	Beginning of period to calculate the aggregate the first time. The size of each period used to calculate the aggregate is defined by the samplingInterval of the MonitoringAttributes (see 7.15).
aggregateType	NodeId	The NodeId of the *HistoryAggregate* object that indicates the list of *Aggregates* to be used when retrieving processed data. See IEC 62541-13 for details.
processingInterval	Duration	The period be used to compute the aggregate.
aggregateConfiguration	Aggregate Configuration	This parameter allows *Clients* to override the *Aggregate* configuration settings supplied by the *AggregateConfiguration* object on a per monitored item basis. See IEC 62541-13 for more information on *Aggregate* configurations. If the *Server* does not support the ability to override the *Aggregate* configuration settings it shall return a *StatusCode* of Bad_AggregateListMismatch.
useSeverCapabilities Defaults	Boolean	If value = TRUE use Aggregate configuration settings as outlined by the AggregateConfiguration object. If value=FALSE use configuration settings as outlined in the following aggregateConfiguration parameters. Default is TRUE.
treatUncertainAsBad	Boolean	As described in IEC 62541-13.
percentDataBad	Byte	As described in IEC 62541-13.
percentDataGood	Byte	As described in IEC 62541-13.
steppedSloped Extrapolation	Boolean	As described in IEC 62541-13.

The AggregateFilterResult defines the revised AggregateFilter the server can return when an AggregateFilter is defined for a MonitoredItem in the CreateMonitoredItems or ModifyMonitoredItems services. The AggregateFilterResult is defined in Table 135.

Table 135 – AggregateFilterResult structure

Name	Type	Description
AggregateFilterResult	structure	
revisedStartTime	UtcTime	The actual StartTime interval that the *Server* shall use. This value is based on a number of factors, including capabilities of the server to access historical data. The revisedStartTime should remain in the same boundary as the startTime (startTime + samplingInterval * n = revisedStartTime).
revisedProcessingInterval	Duration	The actual processingInterval that the *Server* shall use. The revisedProcessingInterval shall be at least twice the revisedSamplingInterval for the MonitoredItem.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.17 MonitoringMode

The *MonitoringMode* is an enumeration that specifies whether sampling and reporting are enabled or disabled for a *MonitoredItem*. The value of the publishing enabled parameter for a *Subscription* does not affect the value of the monitoring mode for a *MonitoredItem* of the *Subscription*. The values of this parameter are defined in Table 136.

Table 136 – MonitoringMode Values

Value	Description
DISABLED_0	The item being monitored is not sampled or evaluated, and *Notifications* are not generated or queued. *Notification* reporting is disabled.
SAMPLING_1	The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is disabled.
REPORTING_2	The item being monitored is sampled and evaluated, and *Notifications* are generated and queued. *Notification* reporting is enabled.

7.18 NodeAttributes parameters

7.18.1 Overview

The *AddNodes Service* allows specifying the *Attributes* for the *Nodes* to add. The *NodeAttributes* is an extensible parameter whose structure depends on the type of the *Attribute* being added. It identifies the *NodeClass* that defines the structure of the *Attributes* that follow. The *parameterTypeIds* are defined in Table 137. The *ExtensibleParamter* type is defined in 7.11.

Table 137 – NodeAttributes parameterTypeIds

Symbolic Id	Description
ObjectAttributes	Defines the *Attributes* for the *Object NodeClass*.
VariableAttributes	Defines the *Attributes* for the *Variable NodeClass*.
MethodAttributes	Defines the *Attributes* for the *Method NodeClass*.
ObjectTypeAttributes	Defines the *Attributes* for the *ObjectType NodeClass*.
VariableTypeAttributes	Defines the *Attributes* for the *VariableType NodeClass*.
ReferenceTypeAttributes	Defines the *Attributes* for the *ReferenceType NodeClass*.
DataTypeAttributes	Defines the *Attributes* for the *DataType NodeClass*.
ViewAttributes	Defines the *Attributes* for the *View NodeClass*.

Table 138 defines the bit mask used in the NodeAttributes parameters to specify which *Attributes* are set by the *Client*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 138 – Bit mask for specified Attributess

Field	Bit	Description
AccessLevel	0	Indicates if the AccessLevel Attribute is set.
ArrayDimensions	1	Indicates if the ArrayDimensions Attribute is set.
Reserved	2	Reserved to be consistent with WriteMask defined in IEC 62541-3.
ContainsNoLoops	3	Indicates if the ContainsNoLoops Attribute is set.
DataType	4	Indicates if the DataType Attribute is set.
Description	5	Indicates if the Description Attribute is set.
DisplayName	6	Indicates if the DisplayName Attribute is set.
EventNotifier	7	Indicates if the EventNotifier Attribute is set.
Executable	8	Indicates if the Executable Attribute is set.
Historizing	9	Indicates if the Historizing Attribute is set.
InverseName	10	Indicates if the InverseName Attribute is set.
IsAbstract	11	Indicates if the IsAbstract Attribute is set.
MinimumSamplingInterval	12	Indicates if the MinimumSamplingInterval Attribute is set.
Reserved	13	Reserved to be consistent with WriteMask defined in IEC 62541-3.
Reserved	14	Reserved to be consistent with WriteMask defined in IEC 62541-3.
Symmetric	15	Indicates if the Symmetric Attribute is set.
UserAccessLevel	16	Indicates if the UserAccessLevel Attribute is set.
UserExecutable	17	Indicates if the UserExecutable Attribute is set.
UserWriteMask	18	Indicates if the UserWriteMask Attribute is set.
ValueRank	19	Indicates if the ValueRank Attribute is set.
WriteMask	20	Indicates if the WriteMask Attribute is set.
Value	21	Indicates if the Value Attribute is set.
Reserved	22:32	Reserved for future use. Shall always be zero.

7.18.2 ObjectAttributes parameter

Table 139 defines the *ObjectAttributes* parameter.

Table 139 – ObjectAttributes

Name	Type	Description
ObjectAttributes	structure	Defines the *Attributes* for the *Object NodeClass.*
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
eventNotifier	Byte	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.3 VariableAttributes parameter

Table 140 defines the *VariableAttributes* parameter.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 140 – VariableAttributes

Name	Type	Description
VariableAttributes	structure	Defines the *Attributes* for the *Variable NodeClass*
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
value	Defined by the *DataType Attribute*	See IEC 62541-3 for the description of this *Attribute*.
dataType	NodeId	See IEC 62541-3 for the description of this *Attribute*.
valueRank	Int32	See IEC 62541-3 for the description of this *Attribute*.
arrayDimensions	UInt32 []	See IEC 62541-3 for the description of this *Attribute*.
accessLevel	Byte	See IEC 62541-3 for the description of this *Attribute*.
userAccessLevel	Byte	See IEC 62541-3 for the description of this *Attribute*.
minimumSamplingInterval	Duration	See IEC 62541-3 for the description of this *Attribute*.
historizing	Boolean	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.4 MethodAttributes parameter

Table 141 defines the *MethodAttributes* parameter.

Table 141 – MethodAttributes

Name	Type	Description
BaseAttributes	structure	Defines the *Attributes* for the *Method NodeClass*
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
executable	Boolean	See IEC 62541-3 for the description of this *Attribute*.
userExecutable	Boolean	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.5 ObjectTypeAttributes parameter

Table 142 defines the *ObjectTypeAttributes* parameter.

Table 142 – ObjectTypeAttributes

Name	Type	Description
ObjectTypeAttributes	structure	Defines the *Attributes* for the *ObjectType NodeClass.*
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
isAbstract	Boolean	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.6 VariableTypeAttributes parameter

Table 143 defines the *VariableTypeAttributes* parameter.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 143 – VariableTypeAttributes

Name	Type	Description
VariableTypeAttributes	structure	Defines the *Attributes* for the *VariableType NodeClass*.
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
value	Defined by the *DataType Attribute*	See IEC 62541-3 for the description of this *Attribute*.
dataType	NodeId	See IEC 62541-3 for the description of this *Attribute*.
valueRank	Int32	See IEC 62541-3 for the description of this *Attribute*.
arrayDimensions	UInt32 []	See IEC 62541-3 for the description of this *Attribute*.
isAbstract	Boolean	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.7 ReferenceTypeAttributes parameter

Table 144 defines the *ReferenceTypeAttributes* parameter.

Table 144 – ReferenceTypeAttributes

Name	Type	Description
ReferenceTypeAttributes	structure	Defines the *Attributes* for the *ReferenceType NodeClass*.
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
isAbstract	Boolean	See IEC 62541-3 for the description of this *Attribute*.
symmetric	Boolean	See IEC 62541-3 for the description of this *Attribute*.
inverseName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.8 DataTypeAttributes parameter

Table 145 defines the *DataTypeAttributes* parameter.

Table 145 – DataTypeAttributes

Name	Type	Description
DataTypeAttributes	structure	Defines the *Attributes* for the *DataType NodeClass*.
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
isAbstract	Boolean	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.18.9 ViewAttributes parameter

Table 146 defines the *ViewAttributes* parameter.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 146 – ViewAttributes

Name	Type	Description
ViewAttributes	structure	Defines the *Attributes* for the *View NodeClass.*
specifiedAttributes	UInt32	A bit mask that indicates which fields contain valid values. A field shall be ignored if the corresponding bit is set to 0. The bit values are defined in Table 138.
displayName	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
description	LocalizedText	See IEC 62541-3 for the description of this *Attribute*.
containsNoLoops	Boolean	See IEC 62541-3 for the description of this *Attribute*.
eventNotifier	Byte	See IEC 62541-3 for the description of this *Attribute*.
writeMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.
userWriteMask	UInt32	See IEC 62541-3 for the description of this *Attribute*.

7.19 NotificationData parameters

7.19.1 Overview

The *NotificationMessage* structure used in the *Subscription Service* set allows to specify different types of *NotificationData*. The *NotificationData* parameter is an extensible parameter whose structure depends on the type of *Notification* being sent. This parameter is defined in Table 147. Other types can be defined by additional parts of this series of standards or other specifications based on OPC UA. The *ExtensibleParamter* type is defined in 7.11.

There may be multiple notifications for a single MonitoredItem in a single NotificationData structure. When that happens the Server shall ensure the notifications appear in the same order that they were queued in the MonitoredItem. Theses notifications do not need to appear as a contiguous block.

Table 147 – NotificationData parameterTypeIds

Symbolic Id	Description
DataChange	*Notification* data parameter used for data change *Notifications*.
Event	*Notification* data parameter used for *Event Notifications*.
StatusChange	*Notification* data parameter used for Subscription status change *Notifications.*

7.19.2 DataChangeNotification parameter

Table 148 defines the *NotificationData* parameter used for data change notifications. This structure contains the monitored data items that are to be reported. Monitored data items are reported under two conditions:

a) if the *MonitoringMode* is set to REPORTING and a change in value or its status (represented by its *StatusCode*) is detected;

b) if the *MonitoringMode* is set to SAMPLING, the *MonitoredItem* is linked to a triggering item and the triggering item triggers.

See 5.12 for a description of the *MonitoredItem Service* set, and in particular the *MonitoringItemModel* and the *TriggeringModel*.

After creating *MonitoredItem* the current value or status of the monitored Attribute shall be queued without applying the filter. If the current value is not available after the first sampling interval the first *Notification* shall be queued after getting the initial value or status from the data source.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 148 – DataChangeNotification

Name	Type	Description
DataChangeNotification	structure	Data change *Notification* data.
monitoredItems []	MonitoredItem Notification	The list of *MonitoredItems* for which a change has been detected.
clientHandle	IntegerId	*Client*-supplied handle for the *MonitoredItem*. The *IntegerId* type is defined in 7.13
value	DataValue	The *StatusCode*, value and timestamp(s) of the monitored *Attribute* depending on the sampling and queuing configuration. If the *StatusCode* indicates an error then the value and timestamp(s) are to be ignored. If not every detected change has been returned since the *Server*'s queue buffer for the *MonitoredItem* reached its limit and had to purge out data, the *Overflow* bit in the *DataValue InfoBits* of the *statusCode* is set. *DataValue* is a common type defined in 7.7.
diagnosticInfos []	DiagnosticInfo	List of diagnostic information. The size and order of this list matches the size and order of the *monitoredItem* parameter. There is one entry in this list for each *Node* contained in the *monitoredItem* parameter. This list is empty if diagnostics information was not requested or is not available for any of the *MonitoredItems*. *DiagnosticInfo* is a common type defined in 7.8.

7.19.3 EventNotificationList parameter

Table 149 defines the *NotificationData* parameter used for *EventNotifications*.

The EventNotificationList defines a table structure that is used to return *Event* fields to a *Client Subscription*. The structure is in the form of a table consisting of one or more *Events*, each containing an array of one or more fields. The selection and order of the fields returned for each *Event* is identical to the selected parameter of the *EventFilter*.

Table 149 – EventNotificationList

Name	Type	Description
EventNotificationList	structure	Event *Notification* data.
events []	EventFieldList	The list of *Events* being delivered.
clientHandle	IntegerId	*Client*-supplied handle for the *MonitoredItem*. The *IntegerId* type is defined in 7.13.
eventFields []	BaseDataType	List of selected *Event* fields. This shall be a one to one match with the fields selected in the *EventFilter*. Subclause 7.16.3 specifies how the *Server* shall deal with error conditions.

7.19.4 StatusChangeNotification parameter

Table 150 defines the *NotificationData* parameter used for a *StatusChangeNotification*.

The *StatusChangeNotification* informs the client about a change in the status of a *Subscription*.

Table 150 – StatusChangeNotification

Name	Type	Description
StatusChangeNotification	structure	Event *Notification* data
status	StatusCode	The *StatusCode* that indicates the status change.
diagnosticInfo	DiagnosticInfo	*DiagnosticInformation* for the status change

7.20 NotificationMessage

The components of this parameter are defined in Table 151.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Table 151 – NotificationMessage

Name	Type	Description
NotificationMessage	structure	The *Message* that contains one or more *Notifications*.
sequenceNumber	Counter	The sequence number of the *NotificationMessage*.
publishTime	UtcTime	The time that this *Message* was sent to the *Client*. If this *Message* is retransmitted to the *Client*, this parameter contains the time it was first transmitted to the *Client*.
notificationData []	Extensible Parameter NotificationData	The list of *NotificationData structures*. The *NotificationData* parameter type is an extensible parameter type specified in 7.19. It specifies the types of *Notifications* that can be sent. The *ExtensibleParameter* type is specified in 7.11. Notifications of the same type should be grouped into one NotificationData element. If a Subscription contains *MonitoredItems* for events and data, this array should have not more than 2 elements. If the *Subscription* contains *MonitoredItems* only for data or only for events, the array size should always be 1 for this *Subscription*.

7.21 NumericRange

This parameter is defined in Table 152. A formal BNF definition of the numeric range can be found in Clause A.3.

The syntax for the string contains one of the following two constructs. The first construct is the string representation of an individual integer. For example, “6” is valid, but “6,0” and “3,2” are not. The minimum and maximum values that can be expressed are defined by the use of this parameter and not by this parameter type definition. The second construct is a range represented by two integers separated by the colon (“:”) character. The first integer shall always have a lower value than the second. For example, “5:7” is valid, while “7:5” and “5:5” are not. The minimum and maximum values that can be expressed by these integers are defined by the use of this parameter, and not by this parameter type definition. No other characters, including white-space characters, are permitted.

Multi-dimensional arrays can be indexed by specifying a range for each dimension separated by a ‘,’. For example, a 2x2 block in a 4x4 matrix could be selected with the range “1:2,0:1”. A single element in a multi-dimensional array can be selected by specifying a single number instead of a range. For example, “1,1” selects the [1,1] element in a two dimensional array.

Dimensions are specified in the order that they appear in the *ArrayDimensions Attribute*. All dimensions shall be specified for a *NumericRange* to be valid.

All indexes start with 0. The maximum value for any index is one less than the length of the dimension.

When reading a value the indexes may not specify a range that is within the bounds of the array. The *Server* shall return a partial result if some elements exist within the range. The *Server* shall return a *Bad_OutOfRange* if no elements exist within the range.

When writing a value the size of the array shall match the size specified by the *NumericRange*. The *Server* shall return an error if it cannot write all elements specified by the *Client*.

The *NumericRange* can also be used to specify substrings for *ByteString* and *String* values. Arrays of *ByteString* and *String* values are treated as two dimensional arrays where the final index specifies the substring range within the *ByteString* or *String* value. The entire *ByteString* or *String* value is selected if the final index is omitted.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 152 – NumericRange

Name	Type	Description
NumericRange	String	A number or a numeric range. A null string indicates that this parameter is not used.

7.22 QueryDataSet

The components of this parameter are defined in Table 153.

Table 153 – QueryDataSet

Name	Type	Description
QueryDataSet	structure	Data related to a *Node* returned in a Query response.
nodeId	ExpandedNodeId	The *NodeId* for this *Node* description.
typeDefinitionNode	ExpandedNodeId	The *NodeId* for the type definition for this *Node* description.
values []	BaseDataType	Values for the selected *Attributes.* The order of returned items matches the order of the requested items. There is an entry for each requested item for the given *TypeDefinitionNode* that matches the selected instance, this includes any related nodes that were specified using a relative path from the selected instance's *TypeDefinitionNode*. If no values where found for a given requested item a null value is returned for that item. If multiple values exist for a requested item then an array of values is returned. If the requested item is a reference then a *ReferenceDescription* or array of *ReferenceDescriptions* are returned for that item.

7.23 ReadValueId

The components of this parameter are defined in Table 154.

Table 154 – ReadValueId

Name	Type	Description
ReadValueId	structure	Identifier for an item to read or to monitor.
nodeId	NodeId	*NodeId* of a *Node*.
attributeId	IntegerId	Id of the *Attribute*. This shall be a valid *Attribute* id. The *IntegerId* is defined in 7.13. The IntegerIds for the Attributes are defined in IEC 62541-6.
indexRange	NumericRange	This parameter is used to identify a single element of an array, or a single range of indexes for arrays. If a range of elements is specified, the values are returned as a composite. The first element is identified by index 0 (zero). The *NumericRange* type is defined in 7.21. This parameter is null if the specified *Attribute* is not an array. However, if the specified *Attribute* is an array, and this parameter is null, then all elements are to be included in the range.
dataEncoding	QualifiedName	This parameter specifies the *BrowseName* of the *DataTypeEncoding* that the *Server* should use when returning the Value *Attribute* of a *Variable*. It is an error to specify this parameter for other *Attributes*. A *Client* can discover what *DataTypeEncodings* are available by following the *HasEncoding Reference* from the *DataType Node* for a *Variable*. OPC UA defines *BrowseNames* which *Servers* shall recognize even if the *DataType Nodes* are not visible in the *Server* address space. These *BrowseNames* are: DefaultBinary The default or native binary (or non-XML) encoding. DefaultXML The default XML encoding. Each *DataType* shall support at least one of these encodings. *DataTypes* that do not have a true binary encoding (e.g. they only have a non-XML text encoding) should use the *DefaultBinary* name to identify the encoding that is considered to be the default non-XML encoding. *DataTypes* that support at least one XML-based encoding shall identify one of the encodings as the DefaultXML encoding. Other standards bodies may define other well-known data encodings that could be supported. If this parameter is not specified then the *Server* shall choose either the DefaultBinary or *DefaultXML* encoding according to what *Message* encoding (see IEC 62541-6) is used for the *Session*. If the *Server* does not support the encoding that matches the *Message* encoding then the *Server* shall choose the default encoding that it does support. If this parameter is specified for a *MonitoredItem*, the *Server* shall set the *StructureChanged* bit in the *StatusCode* (see 7.33) if the *DataTypeEncoding*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Name	Type	Description
		changes. The *DataTypeEncoding* changes if the *DataTypeVersion* of the *DataTypeDescription* or the *DataTypeDictionary* associated with the *DataTypeEncoding* changes.

7.24 ReferenceDescription

The components of this parameter are defined in Table 155.

Table 155 – ReferenceDescription

Name	Type	Description
ReferenceDescription	structure	Reference parameters returned for the *Browse Service*.
referenceTypeId	NodeId	*NodeId* of the *ReferenceType* that defines the *Reference*.
isForward	Boolean	If the value is TRUE, the *Server* followed a forward *Reference*. If the value is FALSE, the *Server* followed an inverse *Reference*.
nodeId	Expanded NodeId	*NodeId* of the *TargetNode* as assigned by the *Server* identified by the *Server* index. The *ExpandedNodeId* type is defined in 7.10. If the *server*Index indicates that the *TargetNode* is a remote *Node*, then the *nodeId* shall contain the absolute namespace URI. If the *TargetNode* is a local *Node* the *nodeId* shall contain the namespace index.
browseName[a)]	QualifiedName	The *BrowseName* of the *TargetNode*.
displayName	LocalizedText	The *DisplayName* of the *TargetNode*.
nodeClass[a)]	NodeClass	*NodeClass* of the *TargetNode*.
typeDefinition[a)]	Expanded NodeId	Type definition *NodeId* of the *TargetNode*. Type definitions are only available for the *NodeClasses Object* and *Variable*. For all other *NodeClasses* a null NodeId shall be returned.
[a)] If the *Server* index indicates that the *TargetNode* is a remote *Node*, then the *browseName*, nodeClass and typeDefinition may be null or empty. If they are not, they might not be up to date because the local *Server* might not continuously monitor the remote *Server* for changes.		

7.25 RelativePath

The components of this parameter are defined in Table 156.

Table 156 – RelativePath

Name	Type	Description
RelativePath	structure	Defines a sequence of *References* and *BrowseNames* to follow.
elements []	RelativePath Element	A sequence of *References* and *BrowseNames* to follow. Each element in the sequence is processed by finding the targets and then using those targets as the starting nodes for the next element. The targets of the final element are the target of the *RelativePath*.
referenceTypeId	NodeId	The type of reference to follow from the current node. The current path cannot be followed any further if the referenceTypeId is not available on the Node instance.
isInverse	Boolean	Indicates whether the inverse *Reference* should be followed. The inverse reference is followed if this value is TRUE.
includeSubtypes	Boolean	Indicates whether subtypes of the *ReferenceType* should be followed. Subtypes are included if this value is TRUE.
targetName	QualifiedName	The *BrowseName* of the target node. The final element may have an empty *targetName*. In this situation all targets of the references identified by the referenceTypeId are the targets of the *RelativePath*. The *targetName* shall be specified for all other elements. The current path cannot be followed any further if no targets with the specified *BrowseName* exist.

A *RelativePath* can be applied to any starting *Node*. The targets of the *RelativePath* are the set of *Nodes* that are found by sequentially following the elements in *RelativePath*.

A text format for the *RelativePath* can be found in Clause A.2. This format is used in examples that explain the *Services* that make use of the *RelativePath* structure.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.26 RequestHeader

The components of this parameter are defined in Table 157.

Table 157 – RequestHeader

Name	Type	Description
RequestHeader	structure	Common parameters for all requests submitted on a *Session*.
authenticationToken	Session AuthenticationToken	The secret *Session* identifier used to verify that the request is associated with the *Session.* The *SessionAuthenticationToken* type is defined in 7.29.
timestamp	UtcTime	The time the *Client* sent the request.
requestHandle	IntegerId	A *requestHandle* associated with the request. This client defined handle can be used to cancel the request. It is also returned in the response.
returnDiagnostics	UInt32	A bit mask that identifies the types of vendor-specific diagnostics to be returned in *diagnosticInfo* response parameters. The value of this parameter may consist of zero, one or more of the following values. No value indicates that diagnostics are not to be returned. Bit Value – Diagnostics to return 0x0000 0001 – ServiceLevel / SymbolicId 0x0000 0002 – ServiceLevel / LocalizedText 0x0000 0004 – ServiceLevel / AdditionalInfo 0x0000 0008 – ServiceLevel / Inner *StatusCode* 0x0000 0010 – ServiceLevel / Inner Diagnostics 0x0000 0020 – OperationLevel / SymbolicId 0x0000 0040 – OperationLevel / LocalizedText 0x0000 0080 – OperationLevel / AdditionalInfo 0x0000 0100 – OperationLevel / Inner *StatusCode* 0x0000 0200 – OperationLevel / Inner Diagnostics Each of these values is composed of two components, *level* and *type*, as described below. If none are requested, as indicated by a 0 value, or if no diagnostic information was encountered in processing of the request, then diagnostics information is not returned. *Level*: ServiceLevel – return diagnostics in the *diagnosticInfo* of the *Service*. OperationLevel – return diagnostics in the *diagnosticInfo* defined for individual operations requested in the *Service*. *Type*: SymbolicId – return a namespace-qualified, symbolic identifier for an error or condition. The maximum length of this identifier is 32 characters. LocalizedText – return up to 256 bytes of localized text that describes the symbolic id. AdditionalInfo – return a byte string that contains additional diagnostic information, such as a memory image. The format of this byte string is vendor-specific, and may depend on the type of error or condition encountered. InnerStatusCode – return the inner *StatusCode* associated with the operation or *Service*. InnerDiagnostics – return the inner diagnostic info associated with the operation or *Service*. The contents of the inner diagnostic info structure are determined by other bits in the mask. Note that setting this bit could cause multiple levels of nested diagnostic info structures to be returned.
auditEntryId	String	An identifier that identifies the *Client*'s security audit log entry associated with this request. An empty string value means that this parameter is not used. The *AuditEntryId* typically contains who initiated the action and from where it was initiated. The *AuditEventId* is included in the *AuditEvent* to allow human readers to correlate an *Event* with the initiating action. More details of the *Audit* mechanisms are defined in 6.2 and in IEC 62541-3.
timeoutHint	UInt32	This timeout in milliseconds is used in the *Client* side *Communication Stack* to set the timeout on a per-call base. For a *Server* this timeout is only a hint and can be used to cancel long running operations to free resources. If the Server detects a timeout, he can cancel the operation by sending the *Service* result *Bad_Timeout*. The *Server* should wait at minimum the timeout after he received the request before cancelling the operation. The value of 0 indicates no timeout.
additionalHeader	Extensible Parameter AdditionalHeader	Reserved for future use. Applications that do not understand the header should ignore it.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.27 ResponseHeader

The components of this parameter are defined in Table 158.

Table 158 – ResponseHeader

Name	Type	Description
ResponseHeader	structure	Common parameters for all responses.
timestamp	UtcTime	The time the *Server* sent the response.
requestHandle	IntegerId	The requestHandle given by the *Client* to the request.
serviceResult	StatusCode	OPC UA-defined result of the *Service* invocation. The *StatusCode* type is defined in 7.33.
serviceDiagnostics	DiagnosticInfo	Diagnostic information for the *Service* invocation. This parameter is empty if diagnostics information was not requested in the request header. The *DiagnosticInfo* type is defined in 7.8.
stringTable []	String	There is one string in this list for each unique namespace, symbolic identifier, and localized text string contained in all of the diagnostics information parameters contained in the response (see 7.8). Each is identified within this table by its zero-based index.
additionalHeader	Extensible Parameter AdditionalHeader	Reserved for future use. Applications that do not understand the header should ignore it.

7.28 ServiceFault

The components of this parameter are defined in Table 159.

The *ServiceFault* parameter is returned instead of the Service response message when a service level error occurs. The *requestHandle* in the *ResponseHeader* should be set to what was provided in the *RequestHeader* even if these values were not valid. The level of diagnostics returned in the *ResponseHeader* is specified by the *returnDiagnostics* parameter in the *RequestHeader*.

The exact use of this parameter depends on the mappings defined in IEC 62541-6.

Table 159 – ServiceFault

Name	Type	Description
ServiceFault	structure	An error response sent when a service level error occurs.
responseHeader	ResponseHeader	Common response parameters (see 7.27 for *ResponseHeader* definition).

7.29 SessionAuthenticationToken

The *SessionAuthenticationToken* type is an opaque identifier that is used to identify requests associated with a particular *Session*. This identifier is used in conjunction with the *SecureChannelId* or *Client Certificate* to authenticate incoming messages. It is the secret form of the *sessionId* for internal use in the *Client* and *Server Applications*.

A Server returns a *SessionAuthenticationToken* in the *CreateSession* response. The *Client* then sends this value with every request which allows the *Server* to verify that the sender of the request is the same as the sender of the original *CreateSession* request.

For the purposes of this discussion, a *Server* consists of application (code) and a *Communication Stack* as shown in Figure 27. The security provided by the *SessionAuthenticationToken* depends on a trust relationship between the *Server* application and the *Communication Stack*. The *Communication Stack* shall be able to verify the sender of the message and it uses the *SecureChannelId* or the *Client Certificate* to identify the sender to the *Server*. In these cases, the *SessionAuthenticationToken* is a UInt32 identifier that allows the *Server* to distinguish between different *Sessions* created by the same sender.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

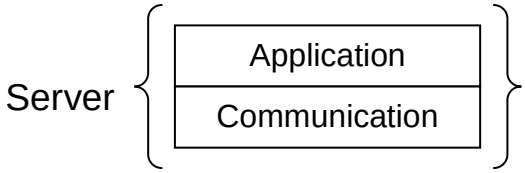


Figure 27 – Logical layers of a *Server*

In some cases, the application and the *Communication Stack* cannot exchange information at runtime which means the application will not have access to the *SecureChannelId* or the *Certificate* used to create the *SecureChannel.* In these cases the application shall create a random *ByteString* value that is at least 32 bytes long. This value shall be kept secret and shall always be exchanged over a *SecureChannel* with encryption enabled. The Administrator is responsible for ensuring that encryption is enabled. The *Profiles* in IEC 62541-7 may define additional requirements for a *ByteString SessionAuthenticationToken.*

Client and *Server* applications should be written to be independent of the *SecureChannel* implementation. Therefore, they should always treat the *SessionAuthenticationToken* as secret information even if it is not required when using some *SecureChannel* implementations.

Figure 28 illustrates the information exchanged between the Client, the Server and the Server *Communication Stack* when the *Client* obtains a *SessionAuthenticationToken.* In Figure 28 the GetSecureChannelInfo step represents an API that depends on the *Communication Stack* implementation.

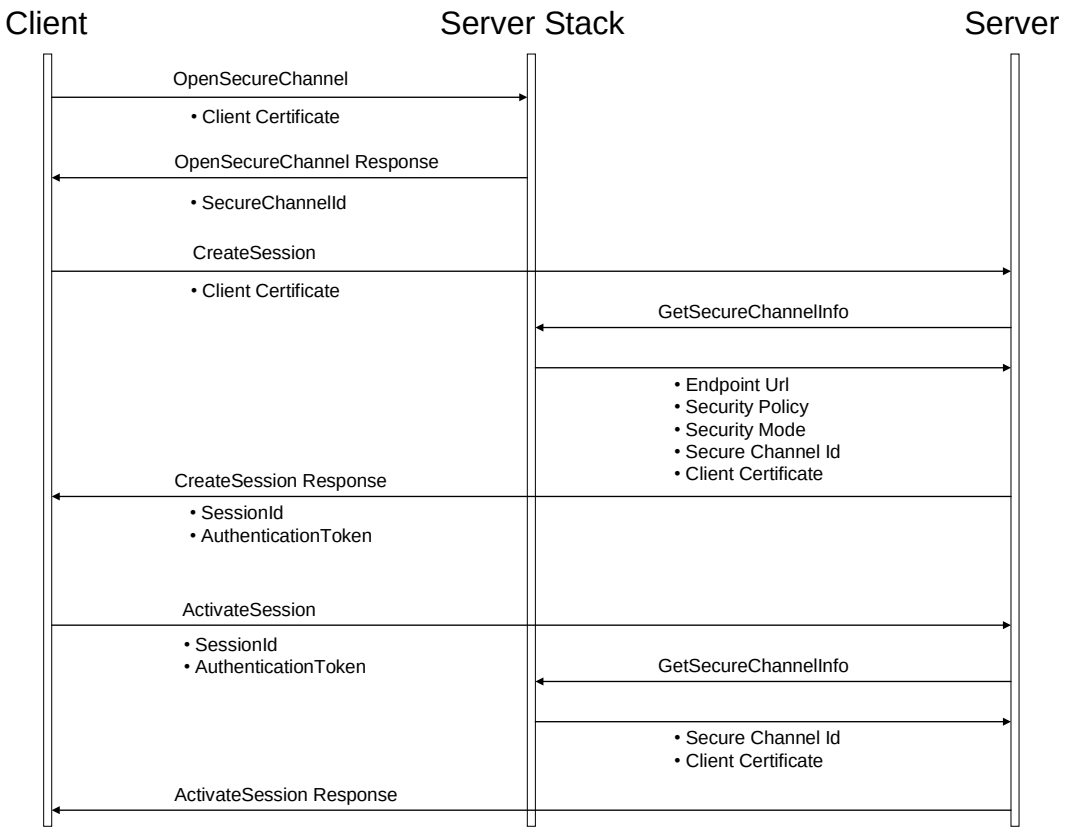


Figure 28 – Obtaining a SessionAuthenticationToken

The *SessionAuthenticationToken* is a subtype of the *NodeId* data type, however, it is never used to identify a *Node* in the address space. *Servers* may assign a value to the *NamespaceIndex*, however, its meaning is *Server* specific.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.30 SignatureData

The components of this parameter are defined in Table 160.

Table 160 – SignatureData

Name	Type	Description
SignatureData	structure	Contains a digital signature created with a *Certificate*.
signature	ByteString	This is a signature generated with the private key associated with a *Certificate*.
agorithm	String	A string containing the URI of the *algorithm*. The URI string values are defined names that may be used as part of the security profiles specified in IEC 62541-7.

7.31 SignedSoftwareCertificate

A *SignedSoftwareCertificate* is a *ByteString* containing an encoded *Certificate.* The encoding of a *SignedSoftwareCertificate* depends on the security technology mapping and is defined completely in IEC 62541-6. Table 161 specifies the information that shall be contained in a *SignedSoftwareCertificate*.

Table 161 – SignedSoftwareCertificate

Name	Type	Description
SignedSoftwareCertificate	structure	A *SoftwareCertificate* with a signature created by Certifying Authority.
version	String	An identifier for the version of the *Certificate* encoding.
serialNumber	ByteString	A unique identifier for the *Certificate* assigned by the Issuer.
signatureAlgorithm	String	The algorithm used to sign the *Certificate*. The syntax of this field depends on the *Certificate* encoding.
signature	ByteString	The signature created by the Issuer.
issuer	Structure	A name that identifies the Issuer *Certificate* used to create the signature.
validFrom	UtcTime	When the *Certificate* becomes valid.
validTo	UtcTime	When the *Certificate* expires.
subject	Structure	A name that identifies the product which the *Certificate* describes. This field shall contain the *productName* and *vendorName* from the *SoftwareCertificate*.
subjectAltName []	Structure	A list of alternate names for the product. This list shall include the *productUri* specified in the *SoftwareCertificate*.
publicKey	ByteString	The public key associated with the *Certificate*.
keyUsage []	String	Specifies how the *Certificate* key may be used. *SignedSoftwareCertificates* may only be used for creating Digital Signatures and Non-Repudiation. The contents of this field depends on the *Certificate* encoding.
softwareCertificate	ByteString	The XML encoded form of the *SoftwareCertificate* stored as UTF8 text. IEC 62541-6 describes the XML representation for the *SoftwareCertificate.*

7.32 SoftwareCertificate

The components of this parameter are defined in Table 162.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Table 162 – SoftwareCertificate

Name	Type	Description
SoftwareCertificate	Structure	A *Certificate* describing a product.
productName	String	The name of the product that is certified. This field shall be specified.
productUri	String	A globally unique identifier for the product that is certified. This field shall be specified.
vendorName	String	The name of the vendor responsible for the product. This field shall be specified.
vendorProductCertificate	ByteString	The DER encoded form of the X.509 *Certificate* which is assigned to the product by the vendor. This field may be omitted.
softwareVersion	String	Software version. This field shall be specified.
buildNumber	String	Build number. This field shall be specified.
buildDate	UtcTime	Date and time of the build. This field shall be specified.
issuedBy	String	URI of the certifying authority. This field shall be specified.
issueDate	UtcTime	Specifies when the *Certificate* was issued by the certifying authority. This field shall be specified.
vendorProductCertificate	ByteString	The DER encoded form of the X.509 *Certificate* which is assigned to the application by the vendor.
supportedProfiles []	structure	List of supported *Profiles.*
organizationUri	String	A URI that identifies the organization that defined the profile.
profileId	String	A string that identifies the *Profile.*
complianceTool	String	A string that identifies the tool or certification method used for compliance testing.
complianceDate	UtcTime	Date and time of the compliance test.
complianceLevel	enum ComplianceLevel	An enumeration that specifies the compliance level of the *Profile*. It has the following values: UNTESTED_0 the profiled capability has not been tested successfully. PARTIAL_1 the profiled capability has been partially tested and has passed critical tests, as defined by the certifying authority. SELFTESTED_2 the profiled capability has been successfully tested using a self-test system authorized by the certifying authority. CERTIFIED_3 the profiled capability has been successfully tested by a testing organisation authorized by the certifying authority.
unsupportedUnitIds []	String	Identifiers for the optional conformance units that were not tested. See IEC 62541-7 for a detailed explanation.

7.33 StatusCode

7.33.1 General

A *StatusCode* in OPC UA is numerical value that is used to report the outcome of an operation performed by an OPC UA *Server*. This code may have associated diagnostic information that describes the status in more detail; however, the code by itself is intended to provide *Client* applications with enough information to make decisions on how to process the results of an OPC UA *Service*.

The *StatusCode* is a 32-bit unsigned integer. The top 16 bits represent the numeric value of the code that shall be used for detecting specific errors or conditions. The bottom 16 bits are bit flags that contain additional information but do not affect the meaning of the *StatusCode*.

All OPC UA *Clients* shall always check the *StatusCode* associated with a result before using it. Results that have an uncertain/warning status associated with them shall be used with care since these results might not be valid in all situations. Results with a bad/failed status shall never be used.

OPC UA *Servers* should return good/success *StatusCodes* if the operation completed normally and the result is always valid. Different *StatusCode* values can provide additional information to the *Client*.

OPC UA *Servers* should use uncertain/warning *StatusCodes* if they could not complete the operation in the manner requested by the *Client*, however, the operation did not fail entirely.

The exact bit assignments are shown in Table 163.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 163 – StatusCode Bit Assignments

Field	Bit Range	Description
Severity	30:31	Indicates whether the *StatusCode* represents a good, bad or uncertain condition. These bits have the following meanings: Good Success 00 Indicates that the operation was successful and the associated results may be used. Uncertain Warning 01 Indicates that the operation was partially successful and that associated results might not be suitable for some purposes. Bad Failure 10 Indicates that the operation failed and any associated results cannot be used. Reserved 11 Reserved for future use. All *Clients* should treat a *StatusCode* with this severity as "Bad".
Reserved	29:28	Reserved for future use. Shall always be zero.
SubCode	16:27	The code is a numeric value assigned to represent different conditions. Each code has a symbolic name and a numeric value. All descriptions in the OPC UA specification refer to the symbolic name. IEC 62541-6 maps the symbolic names onto a numeric value.
StructureChanged	15:15	Indicates that the structure of the associated data value has changed since the last *Notification*. *Clients* should not process the data value unless they re-read the metadata. *Servers* shall set this bit if the *DataTypeEncoding* used for a *Variable* changes.Subclause 7.23 describes how the *DataTypeEncoding* is specified for a *Variable*. The bit is also set if the data type *Attribute* of the *Variable* changes. A *Variable* with data type *BaseDataType* does not require the bit to be set when the data type changes. *Servers* shall also set this bit if the *ArrayDimensions* or the *ValueRank Attribute* or the *EnumStrings Property* of the *DataType* of the *Variable* changes. This bit is provided to warn *Clients* that parse complex data values that their parsing routines could fail because the serialized form of the data value has changed. This bit has meaning only for *StatusCodes* returned as part of a data change *Notification* or the *HistoryRead*. *StatusCodes* used in other contexts shall always set this bit to zero.
SemanticsChanged	14:14	Indicates that the semantics of the associated data value have changed. *Clients* should not process the data value until they re-read the metadata associated with the *Variable*. *Servers* should set this bit if the metadata has changed in a way that could cause application errors if the *Client* does not re-read the metadata. For example, a change to the engineering units could create problems if the *Client* uses the value to perform calculations. IEC 62541-8 defines the conditions where a *Server* shall set this bit for a DA *Variable*. Other specifications may define additional conditions. A *Server* may define other conditions that cause this bit to be set. This bit has meaning only for *StatusCodes* returned as part of a data change *Notification* or the *HistoryRead*. *StatusCodes* used in other contexts shall always set this bit to zero.
Reserved	12:13	Reserved for future use. Shall always be zero.
InfoType	10:11	The type of information contained in the info bits. These bits have the following meanings: NotUsed 00 The info bits are not used and shall be set to zero. DataValue 01 The *StatusCode* and its info bits are associated with a data value returned from the *Server*. Reserved 1X Reserved for future use. The info bits shall be ignored.
InfoBits	0:9	Additional information bits that qualify the *StatusCode*. The structure of these bits depends on the Info Type field.

Table 164 describes the structure of the *InfoBits* when the Info Type is set to *DataValue* (01).

Table 164 – DataValue InfoBits

Info Type	Bit Range	Description
LimitBits	8:9	The limit bits associated with the data value. The limits bits have the following meanings: **Limit** **Bits** **Description** None 00 The value is free to change. Low 01 The value is at the lower limit for the data source. High 10 The value is at the higher limit for the data source. Constant 11 The value is constant and cannot change.
Overflow	7	If this bit is set, not every detected change has been returned since the *Server*'s queue buffer for the *MonitoredItem* reached its limit and had to purge out data.
Reserved	5:6	Reserved for future use. Shall always be zero.
HistorianBits	0:4	These bits are set only when reading historical data. They indicate where the data value came from and provide information that affects how the *Client* uses the data value. The historian bits have the following meaning: Raw XXX00 A raw data value. Calculated XXX01 A data value which was calculated. Interpolated XXX10 A data value which was interpolated. Reserved XXX11 Undefined. Partial XX1XX A data value which was calculated with an incomplete interval.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Info Type	Bit Range	Description
		Extra Data X1XXX A raw data value that hides other data at the same timestamp. Multi Value 1XXXX Multiple values match the aggregate criteria (i.e. multiple minimum values at different timestamps within the same interval). IEC 62541-11 describes how these bits are used in more detail.

7.33.2 Common StatusCodes

Table 165 defines the common *StatusCodes* for all *Service* results. IEC 62541-6 maps the symbolic names to a numeric value.

Table 165 – Common Service Result Codes

Symbolic Id	Description
Good	The operation was successful.
Good_CompletesAsynchronously	The processing will complete asynchronously.
Good_SubscriptionTransferred	The subscription was transferred to another session.
Bad_CertificateHostNameInvalid	The *HostName* used to connect to a *Server* does not match a *HostName* in the *Certificate*.
Bad_CertificateIssuerRevocationUnknown	It was not possible to determine if the Issuer *Certificate* has been revoked.
Bad_CertificateIssuerUseNotAllowed	The Issuer *Certificate* may not be used for the requested operation.
Bad_CertificateIssuerTimeInvalid	An Issuer *Certificate* has expired or is not yet valid.
Bad_CertificateIssuerRevoked	The Issuer *Certificate* has been revoked.
Bad_CertificateInvalid	The certificate provided as a parameter is not valid.
Bad_CertificateRevocationUnknown	It was not possible to determine if the *Certificate* has been revoked.
Bad_CertificateRevoked	The *Certificate* has been revoked.
Bad_CertificateTimeInvalid	The *Certificate* has expired or is not yet valid.
Bad_CertificateUriInvalid	The URI specified in the *ApplicationDescription* does not match the URI in the *Certificate*.
Bad_CertificateUntrusted	The *Certificate* is not trusted.
Bad_CertificateUseNotAllowed	The *Certificate* may not be used for the requested operation.
Bad_CommunicationError	A low level communication error occurred.
Bad_DataTypeIdUnknown	The extension object cannot be (de)serialized because the data type id is not recognized.
Bad_DecodingError	Decoding halted because of invalid data in the stream.
Bad_EncodingError	Encoding halted because of invalid data in the objects being serialized.
Bad_EncodingLimitsExceeded	The message encoding/decoding limits imposed by the *Communication Stack* have been exceeded.
Bad_IdentityTokenInvalid	The user identity token is not valid.
Bad_IdentityTokenRejected	The user identity token is valid but the server has rejected it.
Bad_InternalError	An internal error occurred as a result of a programming or configuration error.
Bad_InvalidArgument	One or more arguments are invalid. Each service defines parameter-specific *StatusCodes* and these *StatusCodes* shall be used instead of this general error code. This error code shall be used only by the *Communication Stack* and in services where it is defined in the list of valid *StatusCodes* for the service.
Bad_InvalidState	The operation cannot be completed because the object is closed, uninitialized or in some other invalid state.
Bad_InvalidTimestamp	The timestamp is outside the range allowed by the server.
Bad_NonceInvalid	The nonce does appear to be not a random value or it is not the correct length.
Bad_NothingToDo	There was nothing to do because the client passed a list of operations with no elements.
Bad_OutOfMemory	Not enough memory to complete the operation.
Bad_RequestCancelledByClient	The request was cancelled by the client.
Bad_RequestTooLarge	The request message size exceeds limits set by the server.
Bad_ResponseTooLarge	The response message size exceeds limits set by the client.
Bad_RequestHeaderInvalid	The header for the request is missing or invalid.
Bad_ResourceUnavailable	An operating system resource is not available.
Bad_SecureChannelIdInvalid	The specified secure channel is not longer valid.
Bad_SecurityChecksFailed	An error occurred verifying security.
Bad_ServerHalted	The server has stopped and cannot process any requests.
Bad_ServerNotConnected	The operation could not complete because the client is not connected to the server.
Bad_ServerUriInvalid	The *Server* URI is not valid.
Bad_ServiceUnsupported	The server does not support the requested service.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Symbolic Id	Description
Bad_SessionIdInvalid	The session id is not valid.
Bad_SessionClosed	The session was closed by the client.
Bad_SessionNotActivated	The session cannot be used because ActivateSession has not been called.
Bad_Shutdown	The operation was cancelled because the application is shutting down.
Bad_SubscriptionIdInvalid	The subscription id is not valid.
Bad_Timeout	The operation timed out.
Bad_TimestampsToReturnInvalid	The timestamps to return parameter is invalid.
Bad_TooManyOperations	The request could not be processed because it specified too many operations.
Bad_UnexpectedError	An unexpected error occurred.
Bad_UnknownResponse	An unrecognized response was received from the server.
Bad_UserAccessDenied	User does not have permission to perform the requested operation.
Bad_ViewIdUnknown	The view id does not refer to a valid view Node.
Bad_ViewTimestampInvalid	The view timestamp is not available or not supported.
Bad_ViewParameterMismatchInvalid	The view parameters are not consistent with each other.
Bad_ViewVersionInvalid	The view version is not available or not supported.

Table 166 defines the common *StatusCodes* for all operation level results. IEC 62541-6 maps the symbolic names to a numeric value. The common *Service* result codes can be also contained in the operation level.

Table 166 – Common Operation Level Result Codes

Symbolic Id	Description
Good_Clamped	The value written was accepted but was clamped.
Good_Overload	Sampling has slowed down due to resource limitations.
Uncertain	The value is uncertain but no specific reason is known.
Bad	The value is bad, but no specific reason is known.
Bad_AttributeIdInvalid	The attribute is not supported for the specified node.
Bad_BrowseDirectionInvalid	The browse direction is not valid.
Bad_BrowseNameInvalid	The browse name is invalid.
Bad_ContentFilterInvalid	The content filter is not valid.
Bad_ContinuationPointInvalid	The continuation point provided is longer valid.
Bad_DataEncodingInvalid	The data encoding is invalid.
Bad_DataEncodingUnsupported	The server does not support the requested data encoding for the node.
Bad_EventFilterInvalid	The event filter is not valid.
Bad_FilterNotAllowed	A monitoring filter cannot be used in combination with the attribute specified.
Bad_FilterOperandInvalid	The operand used in a content filter is not valid.
Bad_HistoryOperationInvalid	The history details parameter is not valid.
Bad_HistoryOperationUnsupported	The server does not support the requested operation.
Bad_IndexRangeInvalid	The syntax of the index range parameter is invalid.
Bad_IndexRangeNoData	No data exists within the range of indexes specified.
Bad_MonitoredItemFilterInvalid	The monitored item filter parameter is not valid.
Bad_MonitoredItemFilterUnsupported	The server does not support the requested monitored item filter.
Bad_MonitoredItemIdInvalid	The monitoring item id does not refer to a valid monitored item.
Bad_MonitoringModeInvalid	The monitoring mode is invalid.
Bad_NoCommunication	Communication with the data source is defined, but not established, and there is no last known value available. This status/sub-status is used for cached values before the first value is received.
Bad_NoContinuationPoints	The operation could not be processed because all continuation points have been allocated.
Bad_NodeClassInvalid	The node class is not valid.
Bad_NodeIdInvalid	The syntax of the node id is not valid.
Bad_NodeIdUnknown	The node id refers to a node that does not exist in the server address space.
Bad_NoDeleteRights	The *Server* will not allow the node to be deleted.
Bad_NodeNotInView	The nodeToBrowse is not part of the view.
Bad_NotFound	A requested item was not found or a search operation ended without success.
Bad_NotImplemented	Requested operation is not implemented.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Symbolic Id	Description
Bad_NotReadable	The access level does not allow reading or subscribing to the *Node*.
Bad_NotSupported	The requested operation is not supported.
Bad_NotWritable	The access level does not allow writing to the *Node*.
Bad_ObjectDeleted	The object cannot be used because it has been deleted.
Bad_OutOfRange	The value was out of range.
Bad_ReferenceTypeIdInvalid	The reference type id does not refer to a valid reference type node.
Bad_SourceNodeIdInvalid	The source node id does not refer to a valid node.
Bad_StructureMissing	A mandadatory structured parameter was missing or null.
Bad_TargetNodeIdInvalid	The target node id does not refer to a valid node.
Bad_TypeDefinitionInvalid	The type definition node id does not reference an appropriate type node.
Bad_TypeMismatch	The value supplied for the attribute is not of the same type as the attribute's value.
Bad_WaitingForInitialData	Waiting for the server to obtain values from the underlying data source. After creating a *MonitoredItem*, it may take some time for the server to actually obtain values for these items. In such cases the server can optionally send a *Notification* with this status prior to the *Notification* with the first valid value.

7.34 TimestampsToReturn

The *TimestampsToReturn* is an enumeration that specifies the *Timestamp Attributes* to be transmitted for *MonitoredItems* or *Nodes* in *HistoryRead*. The values of this parameter are defined in Table 167.

Table 167 – TimestampsToReturn Values

Value	Description
SOURCE_0	Return the source timestamp. If used in *HistoryRead* the source timestamp is used to determine which historical data values are returned.
SERVER_1	Return the *Server* timestamp. If used in *HistoryRead* the *Server* timestamp is used to determine which historical data values are returned.
BOTH_2	Return both the source and *Server* timestamps. If used in *HistoryRead* the source timestamp is used to determine which historical data values are returned.
NEITHER_3	Return neither timestamp. This is the default value for *MonitoredItems* if a *Variable* value is not being accessed. For *HistoryRead* this is not a valid setting.

7.35 UserIdentityToken parameters

7.35.1 Overview

The *UserIdentityToken* structure used in the *Server Service Set* allows *Clients* to specify the identity of the user they are acting on behalf of. The exact mechanism used to identify users depends on the system configuration. The different types of identity tokens are based on the most common mechanisms that are used in systems today. Table 168 defines the current set of user identity tokens. The *ExtensibleParamter* type is defined in 7.11.

Table 168 – UserIdentityToken parameterTypeIds

Symbolic Id	Description
AnonymousIdentityToken	No user information is available.
UserNameIdentityToken	A user identified by user name and password.
X509IdentityToken	A user identified by an X509v3 *Certificate*.
IssuedIdentityToken	A user identified by a WS-*SecurityToken*.

The *Client* shall always prove possession of a *UserIdentityToken* when it passes it to the *Server*. Some tokens include a secret such as a password which the *Server* will accept as proof. In order to protect these secrets the *Token* shall be encrypted before it is passed to the *Server*. Other types of tokens allow the *Client* to create a signature with the secret associated with the *Token*. In these cases, the *Client* proves possession of a *UserIdentityToken* by

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

appending the last *ServerNonce* to the *ServerCertificate* and uses the secret to produce a *Signature* which is passed to the *Server*.

Each *UserIdentityToken* allowed by an *Endpoint* shall have a *UserTokenPolicy* specified in the *EndpointDescription*. The *UserTokenPolicy* specifies what *SecurityPolicy* to use when encrypting or signing. If this *SecurityPolicy* is omitted then the *Client* uses the *SecurityPolicy* in the *EndpointDescription*. If the matching *SecurityPolicy* is set to None then no encryption or signature is required. It is recommended that Applications never set the SecurityPolicy to None for UserTokens that include a secret because these secrets could be used by an attacker to gain access to the system.

Table 169 describes how to serialize *UserIdentityTokens* before applying encryption.

Table 169 – UserIdentityToken Encrypted Token Format

Name	Type	Description
length	Byte[4]	The length of the encrypted data including the *ServerNonce* but excluding the *length* field. This field is a 4 byte unsigned integer encoded with the least significant bytes appearing first.
tokenData	Byte[*]	The token data.
serverNonce	Byte[*]	The last *ServerNonce* returned by the server in the *CreateSession* or *ActivateSession* response.

7.35.2 AnonymousIdentityToken

The *AnonymousIdentityToken* is used to indicate that the Client has no user credentials.

Table 170 defines the AnonymousIdentityToken parameter.

Table 170 – AnonymousIdentityToken

Name	Type	Description
AnonymousIdentityToken	structure	An anonymous user identity.
policyId	String	An identifier for the *UserTokenPolicy* that the token conforms to. The *UserTokenPolicy* structure is defined in 7.36.

7.35.3 UserNameIdentityToken

The *UserNameIdentityToken* is used to pass simple username/password credentials to the *Server*.

This token shall be encrypted if required by the *SecurityPolicy*. The *Server* should specify a *SecurityPolicy* for the *UserTokenPolicy* if the *SecureChannel* has a *SecurityPolicy* of None.

If the token is encrypted the password shall be converted to a UTF8 *ByteString* and then serialized as shown in Table 169.

The *Server* shall decrypt the password and verify the *ServerNonce*.

If the SecurityPolicy is None then the password only contains the UTF-8 encoded password.

Table 171 defines the *UserNameIdentityToken* parameter.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 171 – UserNameIdentityToken

Name	Type	Description
UserNameIdentityToken	structure	UserName value.
policyId	String	An identifier for the *UserTokenPolicy* that the token conforms to. The *UserTokenPolicy* structure is defined in 7.36.
userName	String	A string that identifies the user.
password	ByteString	The password for the user. This parameter shall be encrypted with the Server's Certificate using the algorithm specified by the *SecurityPolicy*.
encryptionAlgorithm	String	A string containing the URI of the *encryptionAlgorithm*. The URI string values are defined names that may be used as part of the security profiles specified in IEC 62541-7. This parameter is null if the password is not encrypted.

7.35.4 X509IdentityTokens

The *X509IdentiyToken* is used to pass an X509v3 *Certificate* which is issued by the user.

This token shall always be accompanied by a signature if required by the *SecurityPolicy*. The *Server* should specify a *SecurityPolicy* for the *UserTokenPolicy* if the *SecureChannel* has a *SecurityPolicy* of None.

Table 172 defines the X509IdentityToken parameter.

Table 172 – X509 Identity Token

Name	Type	Description
X509IdentityToken	structure	X509v3 value.
policyId	String	An identifier for the *UserTokenPolicy* that the token conforms to. The *UserTokenPolicy* structure is defined in 7.36.
certificateData	ByteString	The X509 *Certificate* in DER format.

7.35.5 IssuedIdentityToken

The *IssuedIdentityToken* is used to pass WS-Security compliant *SecurityToken*s to the *Server*.

WS-Security defines a number of token profiles that may be used to represent different types of *SecurityTokens*. For example, Kerberos and SAML tokens have WSS token profiles and shall be exchanged in OPC UA as XML Security Tokens.

The WSS X509 and UserName tokens should not be exchanged as XML security tokens. OPC UA applications should use the appropriate OPC UA identity tokens to pass the information contained in these types of WSS *SecurityTokens.*

These tokens may be encrypted or require a signature. IEC 62541-7 defines profiles that include user related secuity, they also include any requirements for encryption or signing if a signature is required. Additional security profiles specify encryption and signature algorithms.

If the token is encrypted then the XML shall be converted to an UTF8 *ByteString* and then serialized as shown in Table 169.

The *Server* shall decrypt the tokenData and verify the *ServerNonce*.

If the SecurityPolicy is None or if the token only requires signing then the tokenData contains the UTF-8 encoded XML representation of the token.

Table 173 defines the *IssuedIdentityToken* parameter.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table 173 – IssuedIdentityToken

Name	Type	Description
IssuedIdentityToken	structure	WSS value.
policyId	String	An identifier for the *UserTokenPolicy* that the token conforms to. The *UserTokenPolicy* structure is defined in 7.36.
tokenData	ByteString	The XML representation of the token. This parameter may be encrypted with the Server's Certificate.
encryptionAlgorithm	String	A string containing the URI of the *encryptionAlgorithm*. The list of OPC UA-defined names that may be used is specified in IEC 62541-7. This parameter is null if the token is not encrypted.

7.36 UserTokenPolicy

The components of this parameter are defined in Table 174.

Table 174 – UserTokenPolicy

Name	Type	Description
UserTokenPolicy	structure	Specifies a *UserIdentityToken* that a *Server* will accept.
policyId	String	An identifier for the UserTokenPolicy assigned by the Server. The Client specifies this value when it constructs a UserIdentityToken that conforms to the policy. This value is only unique within the context of a single Server.
tokenType	Enum UserIdentity TokenType	The type of user identity token required. This value is an enumeration with one of the following values: ANONYMOUS_0 No token is required. USERNAME_1 A username/password token. CERTIFICATE_2 An X509v3 certificate token. ISSUEDTOKEN_3 Any WS-Security defined token. A tokenType of ANONYMOUS indicates that the *Server* does not require any user identification. In this case, the *Client* application instance *Certificate* is used as the user identification.
issuedTokenType	String	A URI for the type of token. IEC 62541-7 defines URIs for common issued token types. Vendors may specify their own token. This field may only be specified if *TokenType* is ISSUEDTOKEN. WS-Security tokens are sometimes identified by XML QualifiedNames. A URI for the token can be constructed by appending the name to namespace with a ':' separator. The XML QualifiedName can be reconstructed by searching for the last ':' delimiter.
issuerEndpointUrl	String	A optional URL for the token issuing service. The meaning of this value depends on the *issuedTokenType.*
securityPolicyUri	String	The security policy to use when encrypting or signing the *UserToken* when it is passed to the *Server* in the *ActivateSession* request. Subclause 7.35 describes how this parameter is used. The security policy for the SecureChannel is used if this value is omitted.

7.37 ViewDescription

The components of this parameter are defined in Table 175.

Table 175 – ViewDescription

Name	Type	Description
ViewDescription	structure	Specifies a *View.*
viewId	NodeId	*NodeId* of the *View* to *Query*. A null value indicates the entire *AddressSpace.*
timestamp	UtcTime	The time and date desired. The corresponding version is the one with the closest previous creation timestamp. Either the *Timestamp* or the *viewVersion* parameter may be set by a *Client*, but not both. If *ViewVersion* is set this parameter shall be null.
viewVersion	UInt32	The version number for the *View* desired. When *Nodes* are added to or removed from a *View*, the value of a View's *ViewVersion Property* is updated. Either the *Timestamp* or the *viewVersion* parameter may be set by a *Client*, but not both. The ViewVersion *Property* is defined in IEC 62541-3. If *timestamp* is set this parameter shall be 0. The current view is used if timestamp is null and viewVersion is 0.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex A
(informative)

BNF definitions

A.1 Overview over BNF

The BNF (Backus-Naur form) used in this annex uses `<´ and `>´ to mark symbols, `[´ and `]´ to identify optional paths and `|´ to identify alternatives. If the '(' and ')' symbols are used, it indicates sets.

A.2 BNF of RelativePath

A *RelativePath* is a structure that describes a sequence of *References* and *Nodes* to follow. This annex describes a text format for a RelativePath that can be used in documentation or in files used to store configuration information.

The components of a *RelativePath* text format are specified in Table A.1.

Table A.1 – RelativePath

Symbol	Meaning
/	The forward slash character indicates that the *Server* is to follow any subtype of *HierarchicalReferences*.
.	The period (dot) character indicates that the *Server* is to follow any subtype of an *Aggregates ReferenceType*.
<[#!ns:]ReferenceType>	A string delimited by the '<' and '>' symbols specifies the *BrowseName* of a *ReferenceType* to follow. By default, any *References* of the subtypes of the *ReferenceType* are followed as well. A '#' placed in front of the BrowseName indicates that subtypes should not be followed. A '!' in front of the BrowseName is used to indicate that the inverse *Reference* should be followed. The *BrowseName* may be qualified with a namespace index (indicated by a numeric prefix followed by a colon). This namespace index is used to specify the namespace component of the *BrowseName* for the *ReferenceType.* If the namespace prefix is omitted then namespace index 0 is used.
[ns:]BrowseName	A string that follows a '/', '.' or '>' symbol specifies the *BrowseName* of a target *Node* to return or follow. This BrowseName may be prefixed by its namespace index. If the namespace prefix is omitted then namespace index 0 is used. Omitting the final *BrowseName* from a path is equivalent to a wildcard operation that matches all *Nodes* which are the target of the *Reference* specified by the path.
&	The & sign character is the escape character. It is used to specify reserved characters that appear within a *BrowseName*. A reserved character is escaped by inserting the '&' in front of it. Examples of *BrowseNames* with escaped characters are: Received browse path name — Resolves to "&/Name_1" — "/Name_1" "&.Name_2" — ".Name_2" "&:Name_3" — ":Name_3" "&&Name_4" — "&Name_4"

Table A.2 provides examples of *RelativePaths* specified with the text format.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table A.2 – RelativePath Examples

Browse Path	Description
“/2:Block&.Output”	Follows any forward hierarchical *Reference* with target *BrowseName* = “2:Block.Output”.
“/3:Truck.0:NodeVersion”	Follows any forward hierarchical *Reference* with target *BrowseName* = “3:Truck” and from there a forward *Aggregates Reference* to a target with *BrowseName* “0:NodeVersion”.
“<1:ConnectedTo>1:Boiler/1:HeatSensor”	Follows any forward Reference with a *BrowseName* = ‘1:ConnectedTo’ and finds targets with *BrowseName* = ‘1:Boiler’. From there follows any hierarchical *Reference* and find targets with *BrowseName* = ‘1:HeatSensor’.
“<1:ConnectedTo>1:Boiler/”	Follows any forward Reference with a *BrowseName* = ‘1:ConnectedTo’ and finds targets with *BrowseName* = ‘1:Boiler’. From there it finds all targets of hierarchical *References*.
“<0:HasChild>2:Wheel”	Follows any forward Reference with a *BrowseName* = ‘HasChild’ and qualified with the default OPC UA namespace. Then it finds targets with *BrowseName* = ‘Wheel’ qualified with namespace index ‘2’.
“<!HasChild>Truck”	Follows any inverse Reference with a *BrowseName* = ‘HasChild’ (i.e. follows the *HasParent Reference*). Then it finds targets with *BrowseName* = ‘Truck’. In both cases, the namespace component of the *BrowseName* is assumed to be 0.
“<0:HasChild>”	Finds all targets of forward *References* with a *BrowseName* = ‘HasChild’ and qualified with the default OPC UA namespace.

The following BNF describes the syntax of the *RelativePath* text format.

```
<relative-path>  ::= <reference-type> <browse-name> [relative-path]

<reference-type> ::= '/' | '.' | '<' ['#'] ['!'] <browse-name> '>'

<browse-name>    ::= [<namespace-index> ':'] <name>

<namespace-index>      ::= <digit> [<digit>]

<digit>          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |
'8' | '9'

<name>
     ::= (<name-char> | '&' <reserved-char>) [<name>]

<reserved-char>  ::= '/' | '.' | '<' | '>' | ':' | '#' | '!' | '&'

<name-char>      ::=  All valid characters for a String (see IEC 62541-3) excluding
          reserved-chars.
```

A.3 BNF of NumericRange

The following BNF describes the syntax of the NumericRange parameter type.

```
<numeric-range>  ::= <dimension> [',' <dimension>]

<dimension>      ::= <index> [':' <index>]

<index>          ::= <digit> [<digit>]

<digit>          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |
'8' | '9'
```

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex B
(informative)

Content Filter and Query Examples

B.1 Simple ContentFilter examples

B.1.1 Overview

These examples provide fairly simple content filters. A filter similar to these examples may be used in processing events.

The following conventions apply to these examples with regard to how Attribute operands are used (for a definition of this operand see 7.4.4).

- AttributeOperand: Refers to a *Node,* an *Attribute* of a *Node* or the *Value Attribute* of a *Property* associated with a *Node*. In the examples character names of NodeIds are used instead of an actual nodeId, this also applies to Attribute Ids.
- The string representation of relative paths is used instead of the actual structure.
- The NamespaceIndex used in all examples is 12 (it could just as easily have been 4 or 23 or any value). For more information about NamespaceIndex, see IEC 62541-3. The use of the NamespaceIndex illustrates that the information model being used in the examples is not a model defined by this standard, but one created for the examples.

B.1.2 Example 1

For example the logic describe by '(((AType.A = 5) or InList(BType.B, 3,5,7)) and BaseObjectType.displayName LIKE "Main%")' would result in a logic tree as shown in Figure B.1 and a ContentFilter as shown in Table B.1. For this example to return anything, AType and BType both must be subtypes of BaseObjectType, or the resulting "And" operation would always be false.

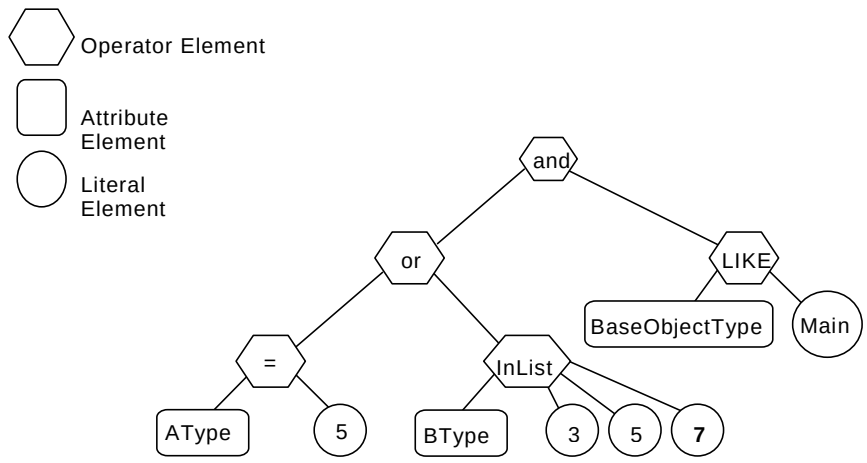


Figure B.1 – Filter Logic Tree Example

Table B.1 describes the elements, operators and operands used in the example.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.1 – ContentFilter Example

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	And	ElementOperand = 1	Element Operand = 4		
1	Or	ElementOperand = 2	Element Operand = 3		
2	Equals	AttributeOperand = NodeId: AType, BrowsePath: ".12:A", AttributeId:value	LiteralOperand = '5'		
3	InList	AttributeOperand = NodeId: BType, BrowsePath: ".12:B", AttributeId:value	LiteralOperand = '3'	LiteralOperand = '5'	LiteralOperand = '7'
4	Like	AttributeOperand = NodeId: BaseObjectType, BrowsePath: ".", AttributeId: displayName	LiteralOperand = "Main%"		

B.1.3 Example 2

As another example, a filter to select all *SystemEvents* (including derived types) that are contained in the Area1 *View* or the Area2 *View* would result in a logic tree, as shown in Figure B.2, and a ContentFilter, as shown in Table B.2.

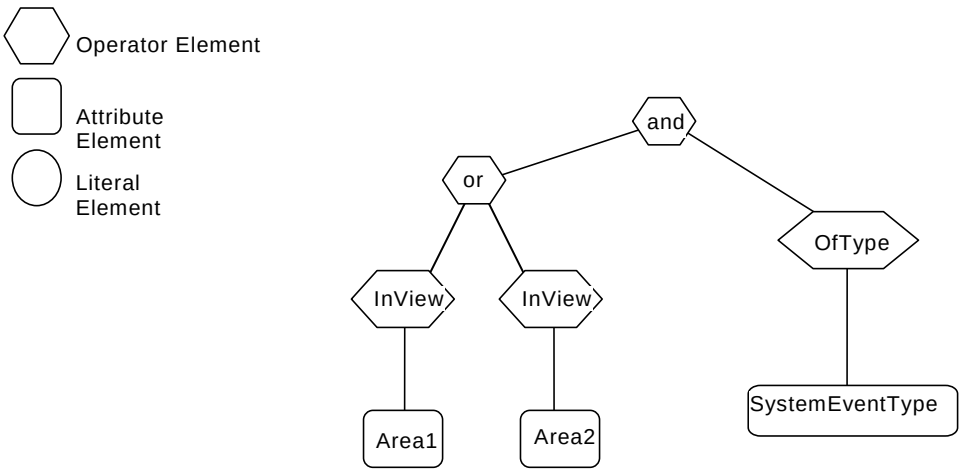


Figure B.2 – Filter Logic Tree Example

Table B.2 describes the elements, operators and operands used in the example.

Table B.2 – ContentFilter Example

Element []	Operator	Operand[0]	Operand[1]
0	And	ElementOperand = 1	ElementOperand = 4
1	Or	ElementOperand = 2	ElementOperand = 3
2	InView	AttributeOperand = NodeId: Area1, BrowsePath: ".", AttributeId: NodeId	
3	InView	AttributeOperand = NodeId: Area2, BrowsePath: ".", AttributeId: NodeId	
4	OfType	AttributeOperand = NodeId: SystemEventType, BrowsePath: ".", AttributeId: NodeId"	

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.2 Complex Examples of ContentFilters (Queries)

B.2.1 Overview

These query examples illustrate complex *ContentFilters.* The following conventions apply to these examples with regard to Attribute operands (for a definition of these operands, see 7.4.4).

- AttributeOperand: Refers to a *Node*, an *Attribute* of a *Node* or the *Value Attribute* of a *Property* associated with a *Node*. In the examples, character names of NodeIds are used instead of an actual nodeId, this also applies to Attribute Ids.
- The string representation of relative paths is used instead of the actual structure.
- The NamespaceIndex used in all examples is 12 (it could just as easily have been 4 or 23 or any value). For more information about NameSpacesIndex, see IEC 62541-3. The use of the NamespaceIndex illustrates that the information model being used in the examples is not a model defined by this standard, but one created for the examples.

B.2.2 Used type model

The following examples use the type model described below.

New Reference types:
“HasChild” derived from HierarchicalReference.
“HasAnimal” derived from HierarchicalReference.
“HasPet” derived from HasAnimal.
“HasFarmAnimal” derived from HasAnimal.
“HasSchedule” derived from HierarchicalReference.

PersonType derived from BaseObjectType adds:
HasProperty "LastName".
HasProperty "FirstName".
HasProperty "StreetAddress".
HasProperty "City".
HasProperty "ZipCode".
May have HasChild reference to a node of type PersonType.
May have HasAnimal reference to a node of type AnimalType (or a subtype of this *Reference* type).

AnimalType derived from BaseObjectType adds:
May have HasSchedule reference to a node of type FeedingScheduleType.
HasProperty "Name".

DogType derived from AnimalType adds:
HasProperty "NickName".
HasProperty "DogBreed".
HasProperty "License".

CatType derived from AnimalType adds:
HasProperty "NickName".
HasProperty "CatBreed".

PigType derived from AnimalType adds:
HasProperty "PigBreed".

ScheduleType derived from BaseObjectType adds:
HasProperty "Period".

FeedingScheduleType derived from ScheduleType adds:
HasProperty "Food".
HasProperty "Amount".

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

AreaType derived from *BaseObjectType* is just a simple *Folder* and contains no *Properties*.

This example type system is shown in Figure B.3. In this Figure, the OPC UA notation is used for all References to *Object* types, *Properties* and subtypes. Additionally, supported *References* are contained in an inner box. The actual references only exist in the instances, thus, no connections to other *Objects* are shown in the Figure and they may be subtypes of the listed *Reference*.

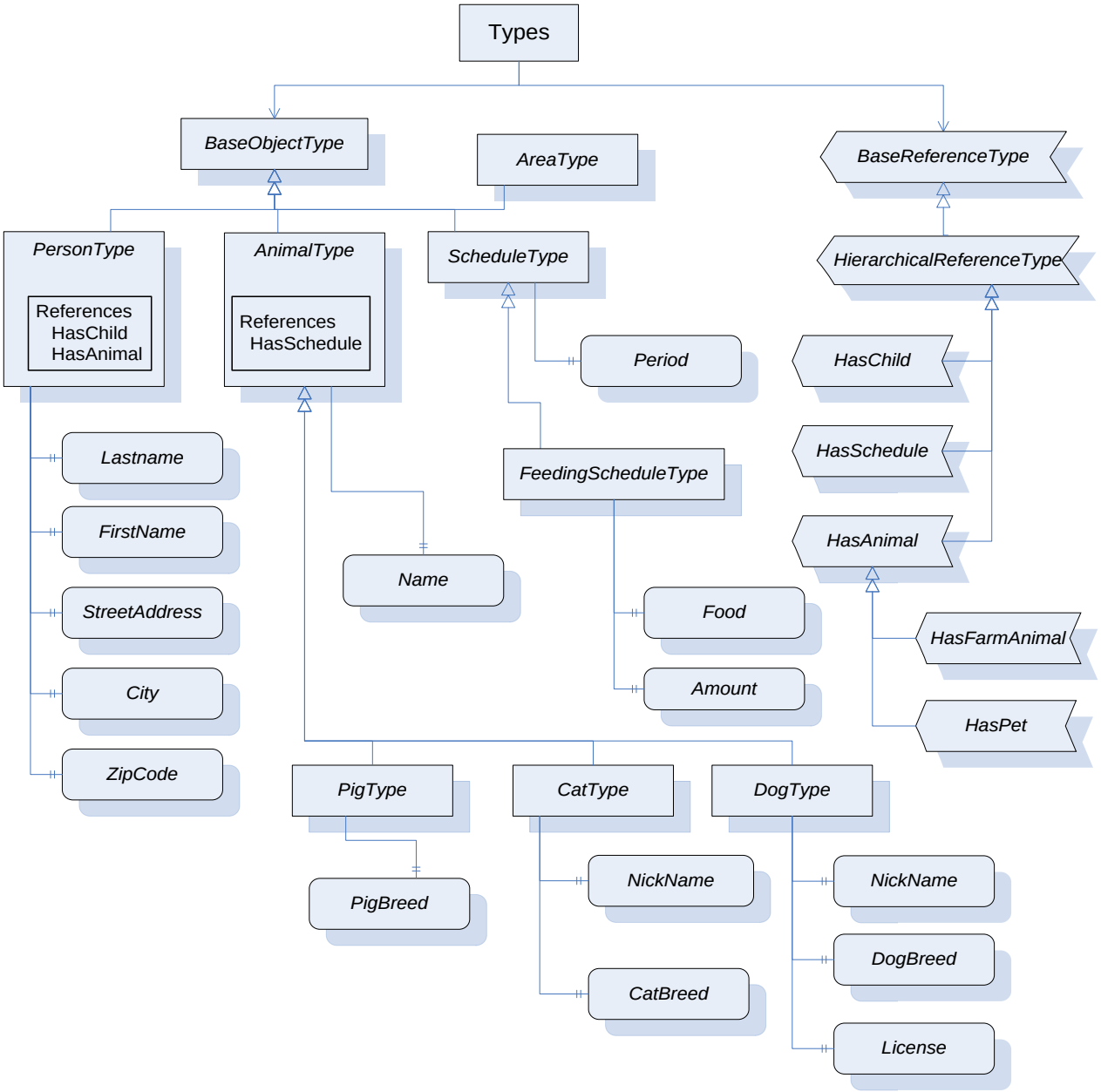


Figure B.3 – Example Type Nodes

A corresponding example set of instances is shown in Figure B.4. These instances include a type *Reference* for *Objects*. Properties also have type *References*, but the *References* are omitted for simplicity. The name of the *Object* is provided in the box and a numeric instance *NodeId* in brackets. Defined *Reference* types use the OPC UA notation, custom *Reference* types are listed with a named *Reference*. For *Properties*, the *BrowseName*, *NodeId*, and *Value* are shown. The *Nodes* that are included in a *View* (View1) are enclosed in the colored

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

box. Two Area nodes are included for grouping of the existing person nodes. All custom nodes are defined in namespace 12 which is not included in Figure B.4.

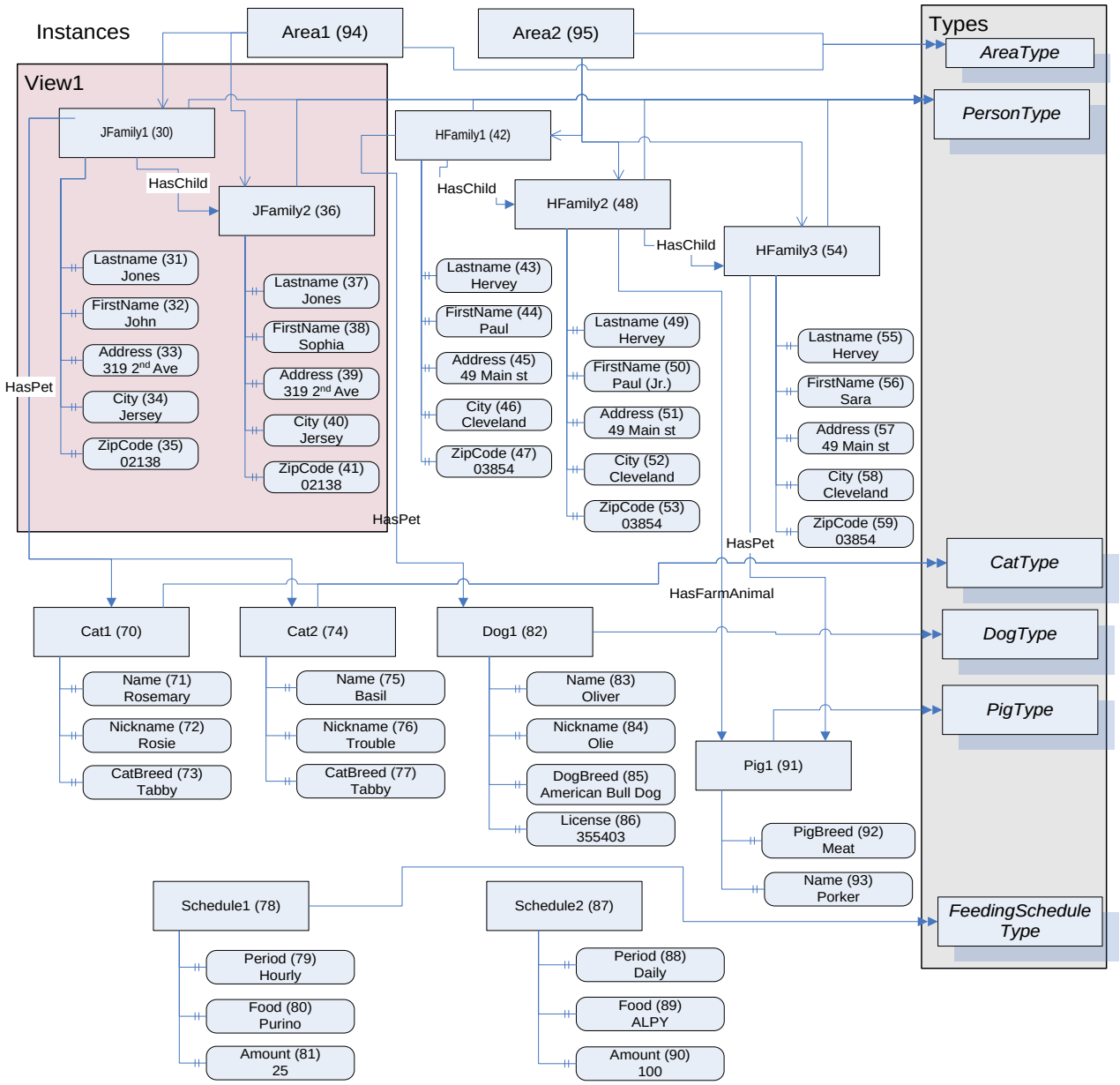


Figure B.4 – Example Instance Nodes

B.2.3 Example Notes

For all of the examples in 7.4.4, the type definition *Node* is listed in its symbolic form, in the actual call it would be the *NodeId* assigned to the *Node*. The *AttributeId* is also the symbolic name of the *Attribute*, in the actual call they would be translated to the *IntegerId* of the *Attribute*. Also, in all of the examples the *BrowseName* is included in the result table for clarity, normally this would not be returned.

The examples assume namespace 12 is the namespace for all of the custom definitions described for the examples.

B.2.4 Example 1

This example requests a simple layered filter, a person has a pet and the pet has a schedule.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Example 1: Get PersonType.lastName, AnimalType.name, ScheduleType.period where the Person Has a Pet and that Pet Has a Schedule.

The *NodeTypeDescription* parameters used in the example are described in Table B.3.

Table B.3 – Example 1 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".12:LastName"	value	N/A
		"<12:HasPet>12:AnimalType. 12:name"	value	N/A
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule. 12:period"	value	N/A

The corresponding *ContentFilter* is illustrated in Figure B.5.

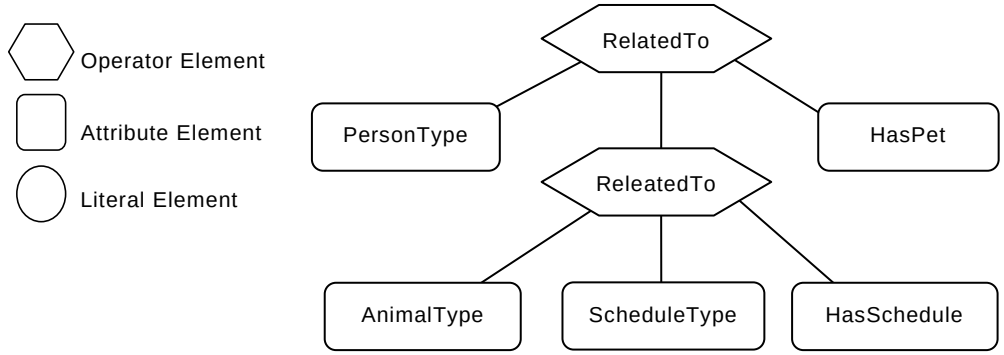


Figure B.5 – Example 1 Filter

Table B.4 describes the *ContentFilter* elements, operators and operands used in the example.

Table B.4 – Example 1 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
1	RelatedTo	AttributeOperand = Nodeid: PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 2	AttributeOperand = NodeId: HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: ScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand= '1'

Table B.5 describes the *QueryDataSet* that results from this query if it were executed against the instances described in Figure B.4

Table B.5 – Example 1 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:30 (JFamily1)	PersonType	".12:lastName"	Jones
		"<12:HasPet>12:AnimalType. 12:name"	Rosemary
			Basil
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule.12:period"	Hourly
			Hourly
12:42(HFamily1)	PersonType	".12:lastName"	Hervey
		"<12:HasPet>12:AnimalType. 12:name"	Olive
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule.12:period"	Daily

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The Value column is returned as an array for each *Node* description, where the order of the items in the array would correspond to the order of the items that were requested for the given Node Type. In addition, if a single *Attribute* has multiple values, then it would be returned as an array within the larger array. For example in this table Rosemary and Basil would be returned in an array the .<hasPet>.AnimalType.name item. They are shown as separate rows for ease of viewing.

NOTE The relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The *TypeDefinition NodeId* would be an integer not the symbolic name that is included in the table.

B.2.5 Example 2

The second example illustrates receiving a list of disjoint *Nodes* and also illustrates that an array of results can be received.

Example 2: Get PersonType.lastName, AnimalType.name where a person has a child or (a pet is of type cat and has a feeding schedule).

The NodeTypeDescription parameters used in the example are described in Table B.6.

Table B.6 – Example 2 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".12:LastName"	Value	N/A
AnimalType	TRUE	".12:name"	Value	N/A

The corresponding ContentFilter is illustrated in

Figure B.6.

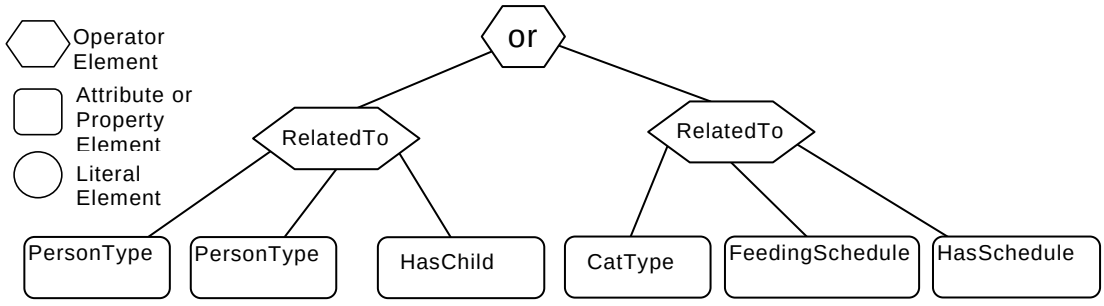


Figure B.6 – Example 2 Filter Logic Tree

Table B.7 describes the elements, operators and operands used in the example. It is worth noting that a Cattype is a subtype of Animaltype.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.7 – Example 2 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

The results from this query would contain the *QueryDataSets* shown in Table B.8.

Table B.8 – Example 2 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:30 (Jfamily1)	Persontype	. 12:lastName	Jones
12:42 (HFamily1)	PersonType	. 12:lastName	Hervey
12:48 (HFamily2)	PersonType	. 12:lastName	Hervey
12:70 (Cat1)	CatType	. 12:name	Rosemary
12:74 (Cat2)	CatType	. 12:name	Basil

NOTE The relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The *TypeDefinitionNodeId* would be a *NodeId* not the symbolic name that is included in the table.

B.2.6 Example 3

The third example provides a more complex *Query* in which the results are filtered on multiple criteria.

Example 3: Get PersonType.lastName, AnimalType.name, ScheduleType.period where a person has a pet and the animal has a feeding schedule and the person has a zipcode = '02138' and the schedule.period is daily or hourly and Amount to feed is > 10.

Table B.9 describes the NodeTypeDescription parameters used in the example.

Table B.9 – Example 3 – NodeTypeDescriptions

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	"12:PersonType.12:lastName"	Value	N/A
		"12:PersonType<12:HasPet>12:AnimalType. 12:name"	Value	N/A
		"12:PersonType<12:HasPet>12:AnimalType<12:HasSchedule> 12:FeedingSchedule.period"	Value	N/A

The corresponding *ContentFilter* is illustrated in Figure B.7.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

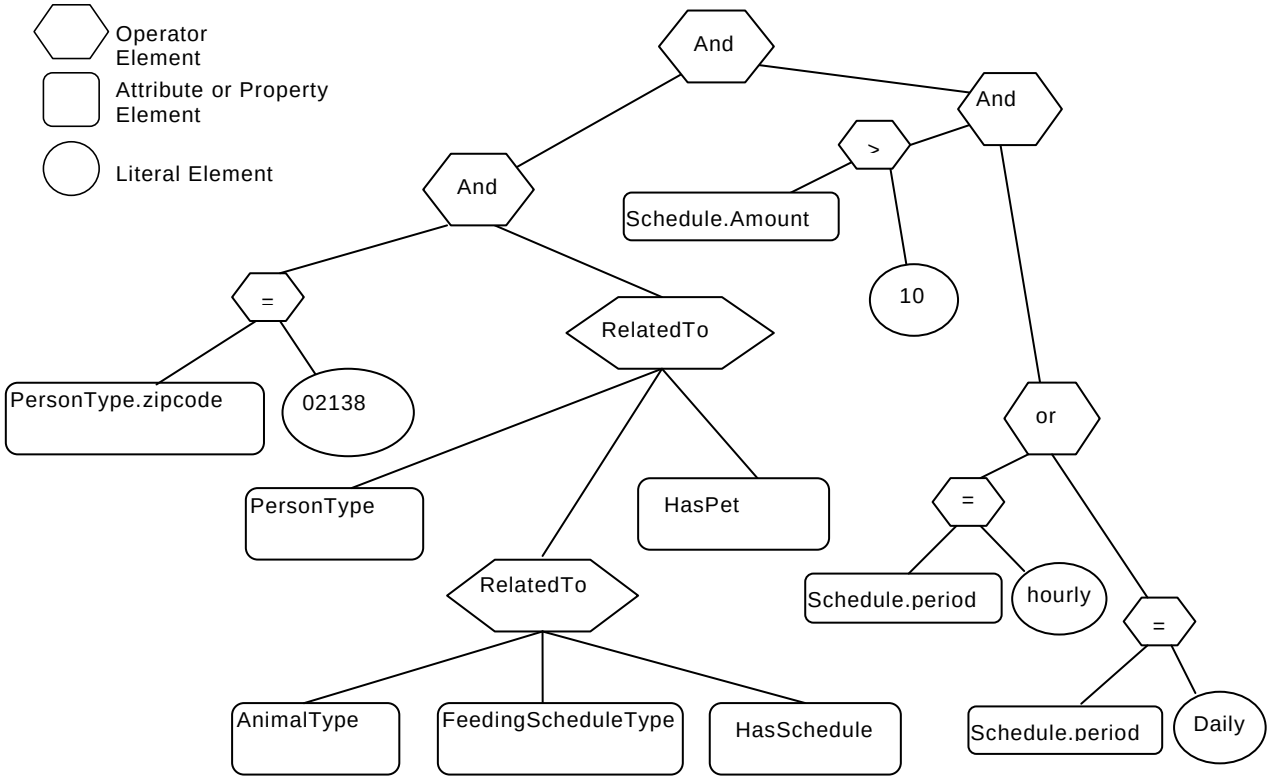


Figure B.7 – Example 3 Filter Logic Tree

Table B.10 describes the elements, operators and operands used in the example.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.10 – Example 3 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	And	Element Operand= 1	ElementOperand = 2		
1	And	ElementOperand = 3	ElementOperand = 5		
2	And	ElementOperand = 3	ElementOperand = 9		
3	Or	ElementOperand = 7	ElementOperand = 8		
4	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 5	AttributeOperand = NodeId: 12:HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
5	RelatedTo	AttributeOperand = Node: 12:AnilmalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
6	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath ".", AttributeId: zipcode	LiteralOperand = '02138'		
7	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Period	LiteralOperand = 'Daily'		
8	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Period	LiteralOperand = 'Hourly'		
9	Greater Than	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Amount	ElementOperand = 10		
10	Cast	LiteralOperand = 10	AttributeOperand = NodeId: String, BrowsePath ".", AttributeId: NodeIdt		

The results from this query would contain the *QueryDataSets* shown in Table B.11.

Table B.11 – Example 3 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:30 (JFamily1)	PersonType	".12:lastName"	Jones
		"<12:hasPet>12:PersonType. 12:name"	Rosemary
			Basil
		"<12:hasPet>12:AnimalType<12:hasSchedule>12:FeedingSchedule. 12:period"	Hourly
			Hourly

NOTE The relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table.

B.2.7 Example 4

The fourth example provides an illustration of the Hop parameter that is part of the RelatedTo Operator.

Example 4: Get PersonType.lastName where a person has a child who has a child who has a pet.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.12 describes the NodeTypeDescription parameters used in the example.

Table B.12 – Example 4 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".12:lastName"	value	N/A

The corresponding *ContentFilter* is illustrated in Figure B.8.

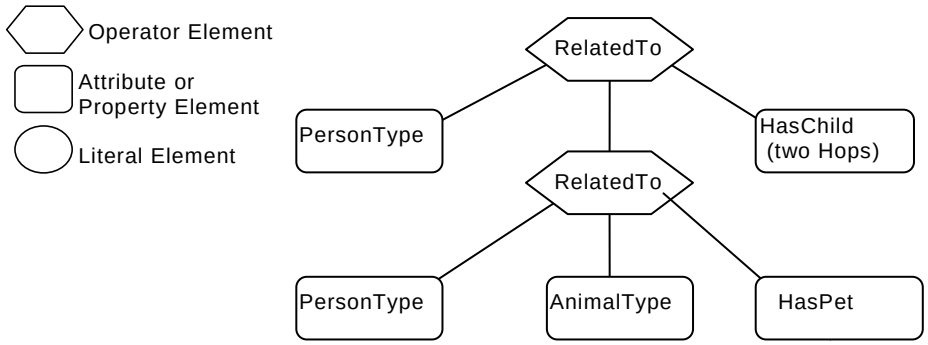


Figure B.8 – Example 4 Filter Logic Tree

Table B.13 describes the elements, operators and operands used in the example.

Table B.13 – Example 4 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	Element Operand = 1	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '2'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

The results from this query would contain the *QueryDataSets* shown in Table B.14. It is worth noting that the pig "Pig1" is referenced as a pet by Sara, but is referenced as a farm animal by Sara's parent Paul.

Table B.14 – Example 4 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:42 (HFamily1)	PersonType	".12:lastName"	Hervey

NOTE The relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table.

B.2.8 Example 5

The fifth example provides an illustration of the use of alias.

Example 5: Get the last names of children that have the same first name as a parent of theirs.

Table B.15 describes the NodeTypeDescription parameters used in the example.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.15 – Example 5 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	“<12:HasChild>12:PersonType. 12:lastName”	Value	N/A

The corresponding *ContentFilter* is illustrated in Figure B.9.

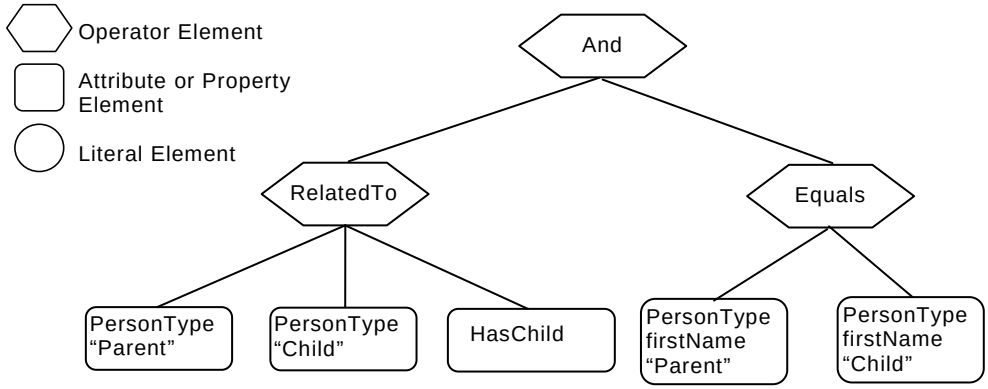


Figure B.9 – Example 5 Filter Logic Tree

In this example, one *Reference* to PersonType is aliased to “Parent” and another *Reference* to PersonType is aliased to “Child”. The value of Parent.firstName and Child.firstName are then compared. Table B.16 describes the elements, operators and operands used in the example.

Table B.16 – Example 5 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3}
0	And	ElementOperand = 1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: NodeId, Alias: “Parent”	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: NodeId, Alias: “Child”	AttributeOperand = NodeId: 12:HasChild, AttributeId: NodeId	LiteralOperand = “1”
2	Equals	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: FirstName, Alias: “Parent”	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: firstName, Alias: “Child”		

The results from this query would contain the *QueryDataSets* shown in Table B.17.

Table B.17 – Example 5 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:42 (HFamily1)	PersonType	“<12:HasChild>12:PersonType.12:lastName”	Hervey

B.2.9 Example 6

The sixth example provides an illustration a different type of request, one in which the *Client* is interested in displaying part of the address space of the server. This request includes listing a *Reference* as something that is to be returned.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Example 6: Get PersonType.NodeId, AnimalType.NodeId, PersonType.HasChild Reference, PersonType.HasAnimal Reference, where a person has a child who has a Animal.

Table B.18 describes the NodeTypeDescription parameters used in the example.

Table B.18 – Example 6 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".12:NodeId"	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>12:AnimalType.NodeId	value	N/A
		<12:HasChild>	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>	value	N/A

The corresponding *ContentFilter* is illustrated in Figure B.10.

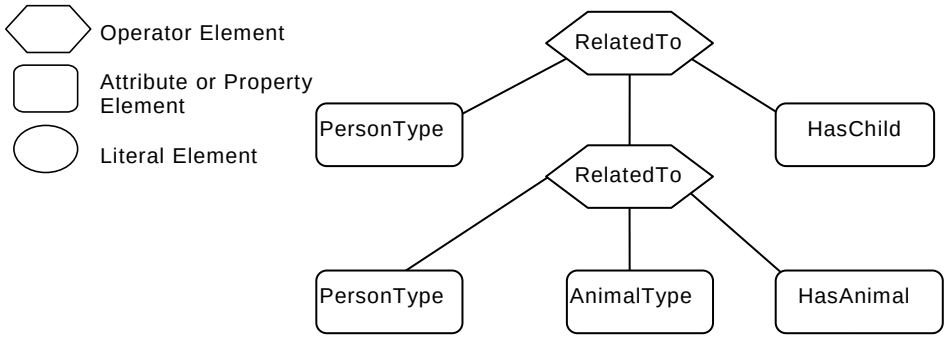


Figure B.10 – Example 6 Filter Logic Tree

Table B.19 describes the elements, operators and operands used in the example.

Table B.19 – Example 6 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 1	AttributeOperand = Node: 12:HasChild, BrowsePath ".",AttributeId:NodeId	LiteralOperand = '1'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasAnimal, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

The results from this query would contain the *QueryDataSets* shown in Table B.20.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.20 – Example 6 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:42 (HFamily1)	PersonType	".NodeId"	12:42 (HFamily1)
		<12:HasChild>12:PersonType<12:HasAnimal> 12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasChild>12:PersonType<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*
12:48 (HFamily2)	PersonType	".NodeId"	12:48 (HFamily2)
		<12:HasChild>12:PersonType<12:HasAnimal> 12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasChild>12:PersonType<12:HasAnimal>	HasPet *ReferenceDescription*

NOTE The relative path and browse name (in parentheses) are not in the *QueryDataSet* and are only shown here for clarity, and the TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table. The value field would in this case be the *NodeId* where it was requested, but for the example the browse name is provided in parentheses and in the case where *Reference* types on the browse name are provided. For the *References* listed in Table B.20, the value would be a *ReferenceDescription* which is described in 7.24.

Table B.21 provides an example of the same QueryDataSet as shown in Table B.20 without any additional fields and minimal symbolic Ids. There is an entry for each requested Attribute, in the cases where an Attribute would return multiple entries the entries are separated by comas. If a structure is being returned then the structure is enclosed in square brackets. In the case of a ReferenceDescription the structure contains a structure and DisplayName and BrowseName are assumed to be the same and defined in Figure B.4.

Table B.21 – Example 6 QueryDataSets without Additional Information

NodeId	TypeDefinition NodeId	Value
12:42	PersonType	12:42
		12:91
		[HasChild,TRUE,[48,HFamily2,HFamily2,PersonType]],
		[HasFarmAnimal,TRUE[91,Pig1,Pig1,PigType]
12:48	PersonType	12:54
		12:91
		[HasChild,TRUE,[54,HFamily3,HFamily3,PersonType]]
		[HasPet, TRUE,[91,Pig1,Pig1,PigType]]

The PersonType, HasChild, PigType, HasPet, HasFarmAnimal identifiers used in Table B.21 would be translated to actual *ExtendedNodeIds*.

B.2.10 Example 7

The seventh example provides an illustration a request in which a *Client* wants to display part of the address space based on a starting point that was obtained via browsing. This request includes listing *References* as something that is to be returned. In this case the Person Browsed to Area2 and wanted to *Query* for information below this starting point.

Example 7: Get PersonType.NodeId, AnimalType.NodeId, PersonType.HasChild Reference, PersonType.HasAnimal Reference where the person is in Area2 (Cleveland nodes) and the person has a child.

Table B.22 describes the NodeTypeDescription parameters used in the example.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.22 – Example 7 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".NodeId"	Value	N/A
		<12:HasChild>	Value	N/A
		<12:HasAnimal>NodeId	Value	N/A
		<12:HasAnimal>	Value	N/A

The corresponding *ContentFilter* is illustrated in

Figure B.11. Note that the *Browse* call would typically return a *NodeId*, thus the first filter is for the *BaseObjectType* with a *NodeId* of 95 where 95 is the *NodeId* associated with the Area2 node, all *Nodes* descend from *BaseObjectType*, and *NodeId* is a base *Property* so this filter will work for all *Queries* of this nature.

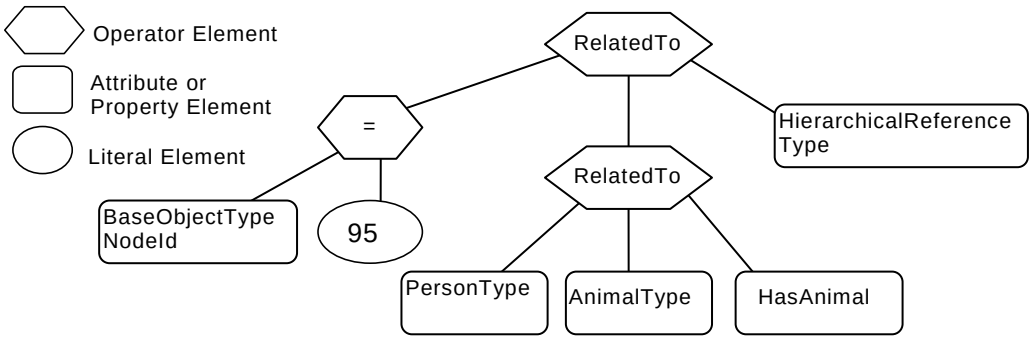


Figure B.11 – Example 7 Filter Logic Tree

Table B.23 describes the elements, operators and operands used in the example.

Table B.23 – Example 7 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: BaseObjectType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 1	AttributeOperand = Node:HierachicalReference, BrowsePath ".", AttributeId:NodeId	LiteralOperand = '1'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonTyp, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	Equals	AttributeOperand = NodeId: BaseObjectType, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '95		

The results from this *Query* would contain the *QueryDataSets* shown in Table B.24.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.24 – Example 7 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:42 (HFamily1)	PersonType	".NodeId"	12:42 (HFamily1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasAnimal>12:AnimalType.NodeId	NULL
		<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*
12:48 (HFamily2)	PersonType	".NodeId"	12:48 (HFamily2)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasAnimal>12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*

NOTE The relative path and browse name (in parentheses) are not in the *QueryDataSet* and are only shown here for clarity, and the TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table. The value field would in this case be the *NodeId* where it was requested, but for the example the browse name is provided in parentheses and in the case where *Reference* types on the browse name are provided. For the *References* listed in Table B.24, the value would be a *ReferenceDescription* which are described in 7.24.

B.2.11 Example 8

The eighth example provides an illustration of a request in which the address space is restricted by a *Server* defined *View*. This request is the same as in the second example which illustrates receiving a list of disjoint *Nodes* and also illustrates that an array of results can be received. It is **important** to note that all of the parameters and the *contentFilter* are the same, only the View description would be specified as "View1".

Example 8: Get PersonType.lastName, AnimalType.name where a person has a child or (a pet is of type cat and has a feeding schedule) limited by the address space in View1.

The NodeTypeDescription parameters used in the example are described in Table B.25.

Table B.25 – Example 8 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".12:LastName"	value	N/A
AnimalType	TRUE	".name"	value	N/A

The corresponding ContentFilter is illustrated in Figure B.12.

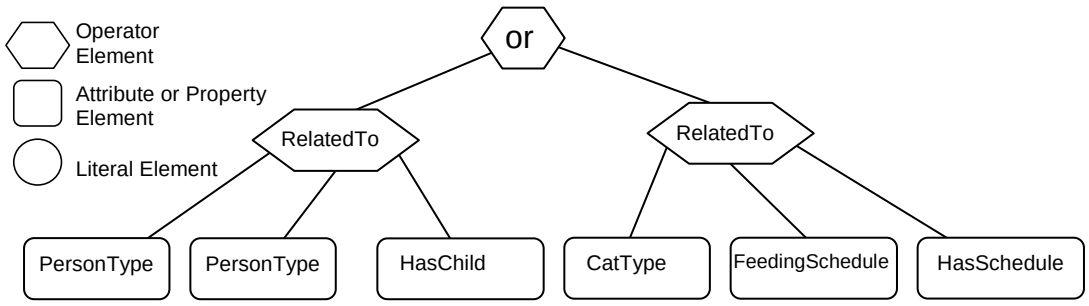


Figure B.12 – Example 8 Filter Logic Tree

Table B.26 describes the elements, operators and operands used in the example. It is worth noting that a CatType is a subtype of AnimalType.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table B.26 – Example 8 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: 12:CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

The results from this query would contain the *QueryDataSets* shown in Table B.27. If this is compared to the result set from example 2, the only difference is the omission of the Cat *Nodes*. These *Nodes* are not in the *View* and thus are not include in the result set.

Table B.27 – Example 8 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:30 (Jfamily1)	Persontype	.12:LastName	Jones

NOTE The relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table.

B.2.12 Example 9

The ninth example provides a further illustration for a request in which the address space is restricted by a *Server* defined *View*. This request is similar to the second example except that some of the requested nodes are expressed in terms of a relative path. It is **important** to note that the *contentFilter* is the same, only the View description would be specified as "View1".

Example 9: Get PersonType.lastName, AnimalType.name where a person has a child or (a pet is of type cat and has a feeding schedule) limited by the address space in View1.

Table B.28 describes the NodeTypeDescription parameters used in the example.

Table B.28 – Example 9 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	Attribute Id	Index Range
PersonType	FALSE	".NodeId"	value	N/A
		<12:HasChild>12:PersonType<12:HasAnimal>12:AnimalType.NodeId	value	N/A
		<12:HasChild>	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>	value	N/A
PersonType	FALSE	".12:LastName"	value	N/A
		<12:HasAnimal>12:AnimalType. 12:Name	value	N/A
AnimalType	TRUE	".12:name"	value	N/A

The corresponding ContentFilter is illustrated in Figure 13.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

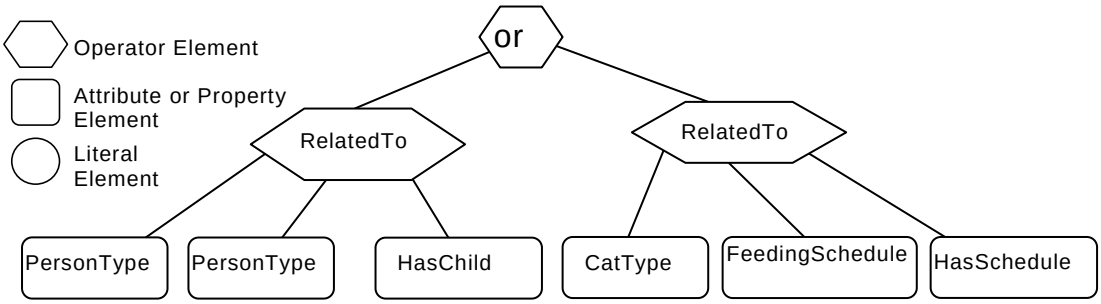


Figure 13 – Example 9 Filter Logic Tree

Table B.29 describes the elements, operators and operands used in the example.

Table B.29 – Example 9 ContentFilter

Element []	Operator	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: 12:CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

The results from this *Query* would contain the *QueryDataSets* shown in Table B.30. If this is compared to the result set from example 2, the Pet *Nodes* are included in the list, even though they are outside of the *View*. This is possible since the name referenced via the relative path and the root *Node* are in the *View*.

Table B.30 – Example 9 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Value
12:30 (Jfamily1)	Persontype	. 12:LastName	Jones
		<12:HasAnimal>12:AnimalType. 12:Name	Rosemary
		<12:HasAnimal>12:AnimalType. 12:Name	Basil

NOTE The the relative path column and browse name (in parentheses in the *NodeId* column) are not in the QueryDataSet and are only shown here for clarity. The TypeDefinitionNodeId would be an integer not the symbolic name that is included in the table.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Bibliography

IEC 62541-11, *OPC Unified Architecture – Part 11: Historical Access*[5]

IEC 62541-12, *OPC Unified Architecture – Part 12: Discovery*[6]

IEC 62541-13, *OPC Unified Architecture – Part 13: Aggregates*[7]

_____

5 Under consideration.

6 Under consideration.

7 Under consideration.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

SOMMAIRE

AVANT-PROPOS 184
INTRODUCTION 186
1 Domaine d'application 187
2 Références normatives 187
3 Termes, définitions et conventions 188
3.1 Termes et définitions 188
3.2 Abréviations 188
3.3 Conventions pour les définitions des Services 189
4 Vue générale 190
4.1 Modèle de jeux de services 190
4.2 Procédures des services Demande/Réponses (Request/Response) 196
5 Jeux de services (Service Sets) 197
5.1 Généralités 197
5.2 En-tête de demande de services et de réponse 197
5.3 Résultats des services 197
5.4 Jeu de services découverte (Discovery Service Set) 198
5.4.1 Vue générale 198
5.4.2 Trouver des serveurs (FindServers) 200
5.4.3 Déterminer les points d'extrémitée (GetEndpoints) 201
5.4.4 Enregistrer un serveur (RegisterServer) 204
5.5 Jeu de services de voie sécurisée (SecureChannel Service Set) 206
5.5.1 Vue d'ensemble 206
5.5.2 Ouvrir un canal sécurisé (OpenSecureChannel) 208
5.5.3 Fermer un canal sécurisé (CloseSecureChannel) 210
5.6 Jeu de services de session (Session Service Set) 211
5.6.1 Vue générale 211
5.6.2 Créer une session (CreateSession) 211
5.6.3 Activer une session (ActivateSession) 215
5.6.4 Fermer une session (CloseSession) 218
5.6.5 Annuler (Cancel) 218
5.7 Jeu de services de gestion de nœuds (NodeManagement Service Set) 219
5.7.1 Vue générale 219
5.7.2 Ajouter des nœuds (AddNodes) 219
5.7.3 Ajouter des références (AddReferences) 221
5.7.4 Supprimer des noeuds (DeleteNodes) 223
5.7.5 Supprimer des références (DeleteReferences) 224
5.8 Jeu de services pour les vues (View Service Set) 225
5.8.1 Vue d'ensemble 225
5.8.2 Parcourir (Browse) 225
5.8.3 Parcourir le prochain (BrowseNext) 227
5.8.4 Traduire les chemins en identifiant de nœuds (TranslateBrowsePathsToNodeIds) 229
5.8.5 Enregistrer les nœuds (RegisterNodes) 231
5.8.6 Supprimer l'enregistrement de nœuds (UnregisterNodes) 232
5.9 Jeu de service d'interrogation (Query Service Set) 233

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.9.1 Vue générale 233
5.9.2 Interroger des vues (Querying Views) 233
5.9.3 Emettre une demande (QueryFirst) 234
5.9.4 Poursuivre la demande (QueryNext) 237
5.10 Jeu de services pour les attributs (Attribute Service Set) 238
5.10.1 Vue générale 238
5.10.2 Lecture (Read) 238
5.10.3 Lecture de l'historique (HistoryRead) 240
5.10.4 Ecrire (Write) 242
5.10.5 Mise à jour de l'historique (HistoryUpdate) 244
5.11 Jeu de service d'appels de fonction (Method Service Set) 245
5.11.1 Vue générale 245
5.11.2 Appel (Call) 246
5.12 Jeu de services des éléments surveillés (MonitoredItem Service Set) 248
5.12.1 Modèle pour les éléments surveillés (MonitoredItem Model) 248
5.12.2 Créer des éléments de controle (CreateMonitoredItems) 253
5.12.3 Modifier des éléments de controle (ModifyMonitoredItems) 256
5.12.4 Gérer des modes de controle (SetMonitoringMode) 258
5.12.5 Gérer des déclenchements (SetTriggering) 259
5.12.6 Supprimer des items (DeleteMonitoredItems) 260
5.13 Subscription Service Set 261
5.13.1 Abonnement (SubscriptionModel) 261
5.13.2 Créer une souscription (CreateSubscription) 269
5.13.3 Modifier une souscription (ModifySubscription) 270
5.13.4 Demander un mode de publication (SetPublishingMode) 271
5.13.5 Publier (Publish) 272
5.13.6 Publier à nouveau (Republish) 274
5.13.7 Transferer une souscription (TransferSubscriptions) 275
5.13.8 Supprimer des souscriptions (DeleteSubscriptions) 277
6 Comportements des services 278
6.1 Sécurité 278
6.1.1 Vue générale 278
6.1.2 Obtenir et installer un certificat d'instance d'application (Application Instance Certificate) 278
6.1.3 Obtenir et installer un Certificat de logiciel 279
6.1.4 Déterminer si un certificat est fiable 281
6.1.5 Valider un Certificat de logiciel 283
6.1.6 Créer un Canal Sécurisé (SecureChannel) 283
6.1.7 Créer une Session 285
6.1.8 Se substituer à un utilisateur 286
6.2 Audits 287
6.2.1 Vue générale 287
6.2.2 Journaux d'audits généraux 287
6.2.3 Événements d'audit généraux 287
6.2.4 Audits pour jeu de services Discovery 288
6.2.5 Audits pour jeu de services SecureChannel 288
6.2.6 Audits pour jeu de services Session 288
6.2.7 Audits pour jeu de services NodeManagement 289
6.2.8 Audits pour jeu de services Attribute 289

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.2.9 Audits pour jeu de services Method 289
6.2.10 Audits pour le View Service Set, le Query Service Set, le MonitoredItem Service Set et le Subscription Service Set 290
6.3 Redondance 290
6.3.1 Vue d'ensemble de la redondance 290
6.3.2 Vue d'ensemble de la redondance du serveur 290
6.3.3 Redondance client 293
7 Définitions de types de paramètres communs 293
7.1 Description d'une application (ApplicationDescription) 293
7.2 Instanciation d'un certificat pour une application (ApplicationInstanceCertificate) 294
7.3 Parcourir les résultats (BrowseResult) 295
7.4 Composantes d'un filtre (ContentFilter) 295
7.4.1 Composantes de la structure d'un filtre (ContentFilter structure) 295
7.4.2 Composantes des résultats d'un filtre (ContentFilterResult) 295
7.4.3 Operateurs d'un filtre (FilterOperator) 296
7.4.4 Paramètre de FilterOperand 303
7.5 Compteur (Counter) 305
7.6 Point de continuation (ContinuationPoint) 305
7.7 Valeur d'une donnée (DataValue) 306
7.7.1 Généralités 306
7.7.2 PicoSeconds 306
7.7.3 SourceTimestamp 306
7.7.4 ServerTimestamp 307
7.7.5 *StatusCode* affecté à une valeur 307
7.8 Information de diagnostic (DiagnosticInfo) 308
7.9 Description d'un point d'extrémité (EndpointDescription) 308
7.10 Identifiant étendu d'un nœud (ExpandedNodeId) 309
7.11 Parametre extensible (ExtensibleParameter) 309
7.12 Indexe (Index) 310
7.13 IntegerId 310
7.14 Mode de sécurité des messages (MessageSecurityMode) 310
7.15 Surveillance des paramètres (MonitoringParameters) 310
7.16 Paramètres des filtres de surveillance (MonitoringFilter) 311
7.16.1 Vue générale 311
7.16.2 DataChangeFilter 312
7.16.3 EventFilter 313
7.16.4 AggregateFilter 315
7.17 Mode de surveillance (MonitoringMode) 316
7.18 Paramètres des attibuts d'un nœud (NodeAttributes) 317
7.18.1 Vue générale 317
7.18.2 Paramètre d'ObjectAttributes 317
7.18.3 Paramètre de VariableAttributes 318
7.18.4 Paramètre de MethodAttributes 318
7.18.5 Paramètre d'ObjectTypeAttributes 318
7.18.6 Paramètre de VariableTypeAttributes 319
7.18.7 Paramètre de ReferenceTypeAttributes 319
7.18.8 Paramètre de DataTypeAttributes 319
7.18.9 Paramètre de ViewAttributes 320

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.19 Paramètres des données de notification (NotificationData) 320
7.19.1 Vue générale 320
7.19.2 Paramètre de DataChangeNotification 320
7.19.3 Paramètre d'EventNotificationList 321
7.19.4 Paramètre de StatusChangeNotification 321
7.20 Message de notification (NotificationMessage) 322
7.21 Plage numérique (NumericRange) 322
7.22 Donnée de demande (QueryDataSet) 323
7.23 Lecture de l'identifiant d'une valeur (ReadValueId) 323
7.24 Description d'une référence (ReferenceDescription) 324
7.25 Chemin relatif (RelativePath) 324
7.26 Entête d'une demande (RequestHeader) 325
7.27 Entête d'une réponse (ResponseHeader) 326
7.28 Service en défaut ServiceFault 326
7.29 Autentification du jeton d'une session (SessionAuthenticationToken) 327
7.30 Signature d'une donnée (SignatureData) 328
7.31 Certificat d'un logiciel signé (SignedSoftwareCertificate) 329
7.32 Certicat d'un logiciel (SoftwareCertificate) 329
7.33 Code d'état (StatusCode) 330
7.33.1 Généralités 330
7.33.2 StatusCodes communs 332
7.34 Marqueurs temporel à retourner (TimestampsToReturn) 335
7.35 Paramètres du jeton identifiant l'utilisateur (UserIdentityToken) 335
7.35.1 Vue générale 335
7.35.2 AnonymousIdentityToken 336
7.35.3 UserNameIdentityToken 336
7.35.4 X509IdentityTokens 337
7.35.5 IssuedIdentityToken 337
7.36 Politique pour le jeton utilisateur (UserTokenPolicy) 338
7.37 Description d'une vue (ViewDescription) 338
Annexe A (informative) Définitions BNF 340
Annexe B (informative) Composition des Filtres et exemples d'interrogation (Query) 342
Bibliographie 360

Figure 1 – Jeu de services de découverte (Discovery Service Set) 191
Figure 2 – Jeu de services de voie sécurisée (SecureChannel Service Set) 191
Figure 3 – Jeu de services de session (Session Service Set) 192
Figure 4 – Jeu de services de gestion de nœuds (NodeManagement Service Set) 192
Figure 5 – Jeu de services de vues (View Service Set) 193
Figure 6 – Jeu de services d'attributs (Attribute Service Set) 194
Figure 7 – Jeu de services de méthodes (Method Service Set) 195
Figure 8 – Jeux de services d'éléments surveillés et d'abonnements (MonitoredItem et Subscription Service Sets) 196
Figure 9 – Processus de découverte 199
Figure 10 – Utilisation d'un Gateway Server (Serveur passerelle) 203
Figure 11 – Processus d'enregistrement – Serveurs lancés manuellement 204
Figure 12 – Processus d'enregistrement – Serveurs lancés automatiquement 205

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Figure 13 – Services SecureChannel et Session 207
Figure 14 – Multiplexage d'utilisateurs sur une Session 213
Figure 15 – Modèle MonitoredItem 249
Figure 16 – Retard type pour détecter des modifications 251
Figure 17 – Modèle de déclenchement 253
Figure 18 – Obtenir et installer un certificat d'instance d'application 279
Figure 19 – Obtention et installation d'un Certificat de logiciel. 280
Figure 20 – Déterminer si un Certificat d'instance d'application est fiable 283
Figure 21 – Etablir un Canal Sécurisé 284
Figure 22 – Etablir une Session 286
Figure 23 – Se substituer à un utilisateur 287
Figure 24 – Configuration type de la redondance transparente 291
Figure 25 – Configuration type de la redondance non transparente 292
Figure 26 – Mode redondance 293
Figure 27 – Couches logiques d'un *Serveur* 327
Figure 28 – Obtenir un SessionAuthenticationToken 328
Figure B.1 – Exemple d'arbre logique de filtre 342
Figure B.2 – Exemple d'arbre logique de filtre 343
Figure B.3 – Exemples de nœuds Type 345
Figure B.4 – Exemple de nœuds Instance 346
Figure B.5 – Filtre de l'exemple 1 347
Figure B.6 – Exemple 2 Arbre logique d'un filtre 349
Figure B.7 – Exemple 3 Arbre logique d'un filtre 350
Figure B.8 – Exemple 4 Arbre logique d'un filtre 352
Figure B.9 – Exemple 5 Arbre logique d'une filtre 353
Figure B.10 – Exemple 6 Arbre logique d'un filtre 354
Figure B.11 – Exemple 7 Arbre logique d'un filtre 356
Figure B.12 – Exemple 8 Arbre logique d'un filtre 357
Figure B.13 – Exemple 9 Arbre logique d'un filtre 359

Tableau 1 – Tableau de définition des services 189
Tableau 2 – Types de paramètre définis dans la CEI 62541-3 190
Tableau 3 – Paramètres du service FindServers 201
Tableau 4 – Paramètres du service GetEndpoints 203
Tableau 5 – Paramètres du service RegisterServer 206
Tableau 6 – Codes des résultats de services RegisterServer 206
Tableau 7 – Paramètres du service OpenSecureChannel 209
Tableau 8 – Codes des résultats de services OpenSecureChannel 210
Tableau 9 – Paramètres du service CloseSecureChannel 210
Tableau 10 – Codes des résultats de services CloseSecureChannel 210
Tableau 11 – Paramètres du service CreateSession 214
Tableau 12 – Codes des résultats de services CreateSession 215
Tableau 13 – Paramètres du service ActivateSession 217

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 14 – Codes des résultats de services ActivateSession 218
Tableau 15 – Paramètres du service CloseSession 218
Tableau 16 – Codes des résultats de services CloseSession 218
Tableau 17 – Paramètres du service Cancel 219
Tableau 18 – Paramètres du service AddNodes 220
Tableau 19 – Codes des résultats de services AddNodes 220
Tableau 20 – Codes de résultats de niveau opération des AddNodes 221
Tableau 21 – Paramètres du service AddReferences 222
Tableau 22 – Codes des résultats de services AddReferences 222
Tableau 23 – Codes de résultats de niveau opération des AddReferences 222
Tableau 24 – Paramètres du service DeleteNodes 223
Tableau 25 – Codes de résultats des services DeleteNodes 223
Tableau 26 – Codes de résultats de niveau opération des DeleteNodes 224
Tableau 27 – Paramètres du service DeleteReferences 224
Tableau 28 – Codes de résultats des services DeleteReferences 225
Tableau 29 – Codes de résultats de niveau opération des DeleteReferences 225
Tableau 30 – Paramètres du service Browse 226
Tableau 31 – Codes de résultats des services Browse 227
Tableau 32 – Codes de résultats de niveau opération de Browse 227
Tableau 33 – Paramètres du service BrowseNext 228
Tableau 34 – Codes de résultats des services BrowseNext 228
Tableau 35 – Codes de résultats de niveau opération de BrowseNext 229
Tableau 36 – Paramètres du service TranslateBrowsePathsToNodeIds 230
Tableau 37 – Codes de résultats des services TranslateBrowsePathsToNodeIds 230
Tableau 38 – Codes de résultats de niveau opération de TranslateBrowsePathsToNodeIds 231
Tableau 39 – Paramètres du service RegisterNodes 232
Tableau 40 – Codes de résultats des services RegisterNodes 232
Tableau 41 – Paramètres du service UnregisterNodes 232
Tableau 42 – Codes de résultats des services UnregisterNodes 233
Tableau 43 – Paramètres de demande QueryFirst 235
Tableau 44 – Paramètres de réponse QueryFirst 236
Tableau 45 – Codes de résultats des services QueryFirst 237
Tableau 46 – Codes de résultats de niveau opération de QueryFirst 237
Tableau 47 – Paramètres du service QueryNext 238
Tableau 48 – Codes de résultats des services QueryNext 238
Tableau 49 – Paramètres du service Read 239
Tableau 50 – Codes de résultats des services Read 240
Tableau 51 – Codes de résultats de niveau opération de Read 240
Tableau 52 – Paramètres du service HistoryRead 241
Tableau 53 – Codes de résultats des services HistoryRead 242
Tableau 54 – Codes de résultats de niveau opération de HistoryRead 242
Tableau 55 – Paramètres du service Write 243

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 56 – Codes de résultats des services Write 243
Tableau 57 – Codes de résultats de niveau opération de Write 244
Tableau 58 – Paramètres du service HistoryUpdate 245
Tableau 59 – Codes de résultats des services HistoryUpdate 245
Tableau 60 – Codes de résultats de niveau opération de HistoryUpdate 245
Tableau 61 – Paramètres du service Call 247
Tableau 62 – Codes de résultats des services Call 247
Tableau 63 – Codes de résultats de niveau opération de Call 248
Tableau 64 – Paramètres du service CreateMonitoredItems 255
Tableau 65 – Codes de résultats des services CreateMonitoredItems 256
Tableau 66 – Codes de résultats de niveau opération de CreateMonitoredItems 256
Tableau 67 – Paramètres du service ModifyMonitoredItems 257
Tableau 68 – Codes de résultats des services ModifyMonitoredItems 257
Tableau 69 – Codes de résultats de niveau opération de ModifyMonitoredItems 258
Tableau 70 – Paramètres du service SetMonitoringMode 258
Tableau 71 – Codes de résultats des services SetMonitoringMode 258
Tableau 72 – Codes de résultats de niveau opération de SetMonitoringMode 259
Tableau 73 – Paramètres du service SetTriggering 259
Tableau 74 – Codes de résultats des services SetTriggering 260
Tableau 75 – Codes de résultats de niveau opération de SetTriggering 260
Tableau 76 – Paramètres du service DeleteMonitoredItems 260
Tableau 77 – Codes de résultats des services DeleteMonitoredItems 261
Tableau 78 – Codes de résultats de niveau opération de DeleteMonitoredItems 261
Tableau 79 – Etats d'un abonnement 264
Tableau 80 – Table d'états d'un abonnement 265
Tableau 81 – Variables d'état et paramètres 267
Tableau 82 – Fonctions 268
Tableau 83 – Paramètres du service CreateSubscription 269
Tableau 84 – Codes de résultats des services CreateSubscription 270
Tableau 85 – Paramètres du service ModifySubscription 270
Tableau 86 – Codes de résultats des services ModifySubscription 271
Tableau 87 – Paramètres du service SetPublishingMode 271
Tableau 88 – Codes de résultats des services SetPublishingMode 272
Tableau 89 – Codes de résultats de niveau opération de SetPublishingMode 272
Tableau 90 – Paramètres du service Publish 273
Tableau 91 – Codes de résultats des services Publish 274
Tableau 92 – Codes de résultats de niveau opération de Publish 274
Tableau 93 – Paramètres du service Republish 274
Tableau 94 – Codes de résultats des services Republish 275
Tableau 95 – Paramètres du service TransferSubscriptions 276
Tableau 96 – Codes de résultats des services TransferSubscriptions 276
Tableau 97 – Codes de résultats de niveau opération de TransferSubscriptions 276
Tableau 98 – Paramètres du service DeleteSubscriptions 277

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 99 – Codes de résultats des services DeleteSubscriptions 277
Tableau 100 – Codes de résultats de niveau opération de DeleteSubscriptions 277
Tableau 101 – Étapes de validation d'un certificat 282
Tableau 102 – Actions de basculement de redondance 292
Tableau 103 – ApplicationDescription 294
Tableau 104 – ApplicationInstanceCertificate 294
Tableau 105 – BrowseResult 295
Tableau 106 – Structure ContentFilter 295
Tableau 107 – Structure ContentFilterResult 296
Tableau 108 – Codes de résultats ContentFilterResult 296
Tableau 109 – Codes de résultats d'opérande ContentFilterResult 296
Tableau 110 – Définition de FilterOperator de base 297
Tableau 111 – Définition de FilterOperator complexe 299
Tableau 112 – Caractères de remplacement 300
Tableau 113 – Règles de conversion 301
Tableau 114 –Règles de préséance des données 302
Tableau 115 – Table de vérité du AND (ET) logique 303
Tableau 116 – Table de vérité du OR (OU) logique 303
Tableau 117 – TypeIds du paramètre FilterOperand 303
Tableau 118 – ElementOperand 304
Tableau 119 – LiteralOperand 304
Tableau 120 – AttributeOperand 304
Tableau 121 – SimpleAttributeOperand 305
Tableau 122 – DataValue 306
Tableau 123 – DiagnosticInfo 308
Tableau 124 – EndpointDescription 309
Tableau 125 – ExpandedNodeId 309
Tableau 126 – Type de base ExtensibleParameter 310
Tableau 127 – Valeurs de MessageSecurityMode 310
Tableau 128 – MonitoringParameters 311
Tableau 129 – TypeIds du paramètre MonitoringFilter 312
Tableau 130 – DataChangeFilter 312
Tableau 131 – Structure EventFilter 314
Tableau 132 – Structure EventFilterResult 314
Tableau 133 – Codes de résultats EventFilterResult 314
Tableau 134 – Structure AggregateFilter 316
Tableau 135 – Structure AggregateFilterResult 316
Tableau 136 – Valeurs de MonitoringMode 316
Tableau 137 –TypeIds du paramètre NodeAttributes 317
Tableau 138 – Masque de bits pour les attributs spécifiés 317
Tableau 139 – ObjectAttributes 318
Tableau 140 – VariableAttributes 318
Tableau 141 – MethodAttributes 318

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 142 – ObjectTypeAttributes 319
Tableau 143 – VariableTypeAttributes 319
Tableau 144 – ReferenceTypeAttributes 319
Tableau 145 – DataTypeAttributes 320
Tableau 146 – ViewAttributes 320
Tableau 147 – TypeIds du paramètre NotificationData 320
Tableau 148 – DataChangeNotification 321
Tableau 149 – EventNotificationList 321
Tableau 150 – StatusChangeNotification 322
Tableau 151 – NotificationMessage 322
Tableau 152 – NumericRange 323
Tableau 153 – QueryDataSet 323
Tableau 154 – ReadValueId 323
Tableau 155 – ReferenceDescription 324
Tableau 156 – RelativePath 325
Tableau 157 – RequestHeader 325
Tableau 158 – ResponseHeader 326
Tableau 159 – ServiceFault 327
Tableau 160 – SignatureData 328
Tableau 161 – SignedSoftwareCertificate 329
Tableau 162 – SoftwareCertificate 330
Tableau 163 – Affectations de bits de StatusCode 331
Tableau 164 – DataValue InfoBits 332
Tableau 165 – Codes de résultats des services communs 333
Tableau 166 – Codes de résultats de niveau opération communs 334
Tableau 167 – Valeurs de TimestampsToReturn 335
Tableau 168 – TypeIds du paramètre UserIdentityToken 335
Tableau 169 – Format de jeton chiffré UserIdentityToken 336
Tableau 170 – AnonymousIdentityToken 336
Tableau 171 – UserNameIdentityToken 337
Tableau 172 – Jeton d'identité X509 337
Tableau 173 – IssuedIdentityToken 338
Tableau 174 – UserTokenPolicy 338
Tableau 175 – ViewDescription 339
Tableau A.1 – RelativePath 340
Tableau A.2 – Exemples de RelativePath 341
Tableau B.1 – Exemple de ContentFilter 343
Tableau B.2 – Exemple de ContentFilter 343
Tableau B.3 – Exemple 1 NodeTypeDescription 347
Tableau B.4 – ContentFiltre de l'exemple 1 ContentFilter 347
Tableau B.5 – Exemple 1 QueryDataSets 348
Tableau B.6 – Exemple 2 NodeTypeDescription 348
Tableau B.7 – Exemple 2 ContentFilter 349

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau B.8 – Exemple 2 QueryDataSets 349
Tableau B.9 – Exemple 3 – NodeTypeDescriptions 350
Tableau B.10 – Exemple 3 ContentFilter 351
Tableau B.11 – Exemple 3 QueryDataSets 351
Tableau B.12 – Exemple 4 NodeTypeDescription 352
Tableau B.13 – Exemple 4 ContentFilter 352
Tableau B.14 – Exemple 4 QueryDataSets 352
Tableau B.15 – Exemple 5 NodeTypeDescription 353
Tableau B.16 – Exemple 5 ContentFilter 353
Tableau B.17 – Exemple 5 QueryDataSets 353
Tableau B.18 – Exemple 6 NodeTypeDescription 354
Tableau B.19 – Exemple 6 ContentFilter 354
Tableau B.20 – Exemple 6 QueryDataSets 355
Tableau B.21 – Exemple 6 QueryDataSets sans informations supplémentaires 355
Tableau B.22 – Exemple 7 NodeTypeDescription 356
Tableau B.23 – Exemple 7 ContentFilter 356
Tableau B.24 – Exemple 7 QueryDataSets 356
Tableau B.25 – Exemple 8 NodeTypeDescription 357
Tableau B.26 – Exemple 8 ContentFilter 357
Tableau B.27 – Exemple 8 QueryDataSets 358
Tableau B.28 – Exemple 9 NodeTypeDescription 358
Tableau B.29 – Exemple 9 ContentFilter 359
Tableau B.30 – Exemple 9 QueryDataSets 359

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 4: Services

AVANT-PROPOS

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

La Norme internationale CEI 62541-4 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité technique 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65E/191/FDIS	65E/213/RVD

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

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Une liste de toutes les parties de la série CEI 62541, sous le titre général *Architecture Unifiée OPC*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

INTRODUCTION

La présente Norme internationale est une spécification destinée aux développeurs d'applications OPC UA. La spécification est le résultat d'un processus d'analyse et de conception visant à développer une interface normalisée qui facilite l'interopérabilité d'applications issues de divers fournisseurs.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

ARCHITECTURE UNIFIÉE OPC –

Partie 4: Services

1 Domaine d'application

La présente partie de la CEI 62541 définit les Services de l'Architecture unifiée OPC (OPC UA). Les *Services* décrits sont le recueil d'appels de procédures abstraites distantes (RPC Remote Procedure Calls) qui sont mises en œuvre par les *Serveur*s OPC UA et qui sont appelées par les Clients OPC UA. Toutes les interactions entre *Clients* et *Serveur*s OPC UA ont lieu via ces Services. Les Services définis sont considérés abstraits car pour leur mise en œuvre aucun mécanisme RPC particulier n'a été spécifié dans cette partie de la CEI 62541. La CEI 62541-6 spécifie les mécanismes de mise en correspondance (mapping). Par exemple, dans la CEI 62541-6 un des mécanismes de mise en correspondance est basé sur l'utilisation des Services Web XML. Dans ce cas, les Services décrits dans la présente partie de la CEI 62541 apparaissent comme les méthodes de service web dans le contrat WSDL.

Tous les *Serveur*s OPC UA n'auront pas besoin de mettre en œuvre la totalité des Services définis. La CEI 62541-7 définit les *Profiles* (profils) qui dictent les Services qui auront besoin d'être mis en œuvre afin d'être conformes à un *Profile* particulier.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI/TR 62541-1:2010, *Specification OPC UA – Part 1: Overview and Concepts* (disponible en anglais uniquement)

CEI/TR 62541-2, *Specification OPC UA – Part-2: Security Model* (disponible en anglais uniquement)

CEI 62541-3, *Specification OPC UA – Part 3: Address Space Model* (disponible en anglais uniquement)

CEI 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'Information*[1]

CEI 62541-6, *Architecture unifiée OPC – Partie 6: Correspondances*[2]

CEI 62541-7, *Architecture unifiée OPC – Partie 7: Profils*[3]

CEI 62541-8, *Architecture unifiée OPC – Partie 8: Accès aux données*[4]

---

1 A publier.

2 A publier.

3 A publier.

4 A publier.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

ISO/CEI 7498 (toutes les parties), *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base*

http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

3 Termes, définitions et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants ainsi que les termes et définitions donnés dans la CEI 62541-1, la CEI 62541-2 et la CEI 62541-3 s'appliquent.

3.1.1
bande morte (Deadband)
plage permise pour les variations de valeur qui ne déclencheront pas une *Notification* de changement de données

NOTE *Bande morte* peut s'appliquer comme un filtre lors de l'abonnement à des *Variables* et sert à empêcher que des signaux bruyants ne mettent inutilement un Client à jour. La présente norme définit *AbsoluteDeadband* comme un filtre commun. La CEI 62541-8 définit un filtre *Deadband* supplémentaire.

3.1.2
point d'extrémité (Endpoint)
adresse physique disponible sur un réseau qui permet à des Clients d'accéder à un ou plusieurs Services fournis par un *Server* (serveur)

NOTE Chaque *Serveur* peut avoir plusieurs *points d'extrémité*. Il faut que l'adresse d'un *point d'extrémité* inclue un *HostName* (un nom d'hôte).

3.1.3
serveur passerelle (Gateway Server)
serveur qui agit comme un intermédiaire pour un ou plusieurs *Serveurs*

NOTE Les *serveurs passerelles* peuvent être déployés pour limiter l'accès interne, assurer la conversion de protocole ou fournir des caractéristiques que les *Serveurs* sous-jacents ne prennent pas en charge.

3.1.4
nom d'hôte (HostName)
identificateur unique pour une machine sur un réseau

NOTE Cet identificateur doit être unique dans un réseau local, mais il peut toutefois être également unique au niveau global.

3.1.5
Jeton de sécurité (Security Token)
identificateur pour un jeu de clés cryptographiques

NOTE Tous les *Security Tokens* (jetons de sécurité) appartiennent à un contexte de sécurité qui est, dans le cas d'OPC UA, la voie sécurisée (*Secure Channel*).

3.1.6
Certificat logiciel (SoftwareCertificate)
certificat numérique pour un produit logiciel qui peut être installé sur plusieurs hôtes pour décrire les fonctions du produit logiciel

NOTE Des installations différentes du même produit logiciel pourraient avoir le même certificat de logiciel.

3.2 Abréviations

API Application Programming Interface (Interface de programmation d'application)

BNF Backus-Naur Form (Notation Backus-Naur)

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

CA Certificate Authority (Autorité de certification)

CRL Certificate Revocation List (Liste d'annulation de certificats)

CTL Certificate Trust List (Liste de certificats de confiance)

DA Data Access (Accès aux données)

UA Unified Architecture (Architecture Unifiée)

URI Uniform Resource Identifier (Identificateur uniforme de ressource)

URL Uniform Resource Locator (Localisateur uniforme de ressource)

3.3 Conventions pour les définitions des Services

Les *Services* OPC UA ont des paramètres qui sont acheminés entre le *Client* et le *Serveur*. Les spécifications des *Services* OPC UA utilisent des tableaux pour décrire les paramètres des *Services* comme montré au Tableau 1. Dans ce tableau, les paramètres sont organisés en paramètres de demande et paramètres de réponse.

Tableau 1 – Tableau de définition des services

Nom	Type	Description
Request (Demande)		Définit les paramètres de demande relatifs au *Service*
Nom de paramètre simple		Description de ce paramètre
Nom de paramètre construit		Description du paramètre construit
Nom de paramètre composant		Description du paramètre composant
Response (Réponse)		Définit les paramètres de réponse relatifs au *Service*

Les colonnes Nom, Type et Description donnent le nom, le type de donnée et la description de chaque paramètre. Tous les paramètres sont obligatoires, même si certains peuvent être inutilisés dans certaines circonstances. La colonne Description spécifie la valeur à fournir lorsqu'un paramètre est inutilisé.

Deux types de paramètres sont définis dans ces tableaux, à savoir simple et construit. Les paramètres simples ont un type de donnée simple, tel que *Boolean* (booléen) ou *String* (chaîne).

Les paramètres construits sont des paramètres composés de deux ou plusieurs paramètres, qui peuvent être simple ou construits. Les noms des paramètres composants sont indentés sous le nom du paramètre construit.

Les types de données utilisés dans ces tableaux peuvent être des types de base, des types communs à plusieurs *Services* ou des types spécifiques à chaque *Service*. Les types de données de base sont définis dans la CEI 62541-3. Les types de base utilisés dans les *Services* sont énumérés dans le Tableau 2. Les types de données qui sont communs à plusieurs Services sont définis à l'Article 7. Les types de données qui sont spécifiques à un Service sont définis dans le tableau de paramètres du *Service* en question.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 2 – Types de paramètre définis dans la CEI 62541-3

Type du paramètre
BaseDataType
NodeId
QualifiedName
LocaleId
Boolean
ByteString
Double
Guid
Int32
String
UInt16
UInt32
UInteger
UTCTime
XmlElement
Duration

Le terme *Services* utilisé dans la présente norme est cohérent avec la définition donnée dans l'ISO/CEI 7498. Les paramètres des primitives du service Request et Indication sont représentés au Tableau 1 comme des paramètres Request. De même, les paramètres des primitives du service Response et Confirmation sont représentés au Tableau 1 comme des paramètres Response. Tous les paramètres de demande et de réponse sont inchangés et acheminés entre l'expéditeur et le récipiendaire. Par conséquent, des colonnes distinctes pour les valeurs des paramètres de demande, d'indication, de réponse et de confirmation sont inutiles et ont été intentionnellement omises afin d'améliorer la lisibilité.

4 Vue générale

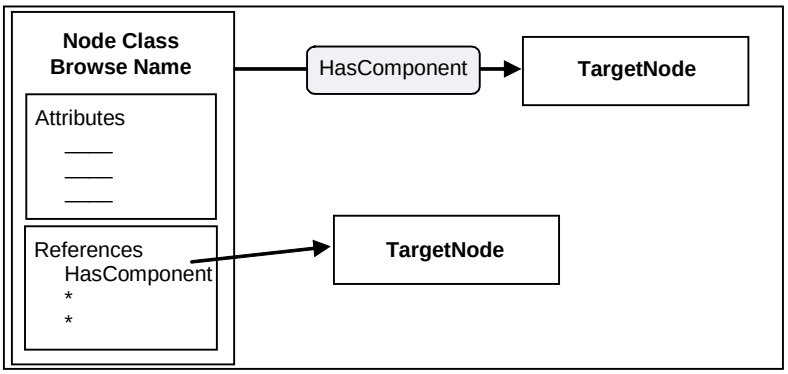
4.1 Modèle de jeux de services

Le présent article spécifie les *Services* OPC UA. Les définitions des *Services* OPC UA sont des descriptions abstraites mais ne sont pas une spécification de mise en œuvre (implémentation). La mise en correspondance entre les descriptions abstraites et la *Pile de communication* (*Communication Stack)* associée à ces *Services* sont définies dans la CEI 62541-6. Dans le cas d'une implémentation basée sur des services web, les *Services* OPC UA correspondent au service web et un *Service* OPC UA correspond à une opération du service web.

Ces *Services* sont organisés en *Service Sets* (jeux de services). Chaque *Service Set* définit un jeu de *Services* connexes. L'organisation en *Service Sets* est un regroupement logique présenté dans cette norme mais qui n'est pas utilisée pour l'implémentation.

Le *Discovery Service Set* (Jeu de services de découverte), illustré à la Figure 1, définit les *Services* qui permettent à un *Client* de découvrir les *Endpoints* implémentés par un *Serveur* et de connaitre le niveau de sécurité de chacun de ces *Endpoints*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

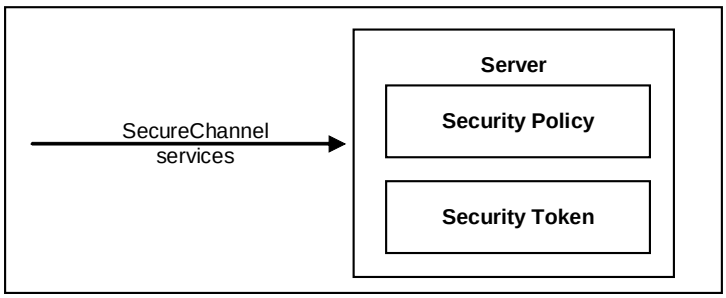


Légende

Anglais	Français
Node Class	Classe de nœuds
Browse Name	Nom de navigation
Attributes	Attributs
References	Références

Figure 1 – Jeu de services de découverte (Discovery Service Set)

Le *SecureChannel Service Set* (Jeu de services de voie sécurisée), illustré à la Figure 2, définit les *Services* qui permettent à un *Client* d'établir unevoie de communication pour assurer la *Confidentiality* (Confidentialité) et l'*Integrity* (Intégrité) des *Messages* échangés avec le *Serveur*.



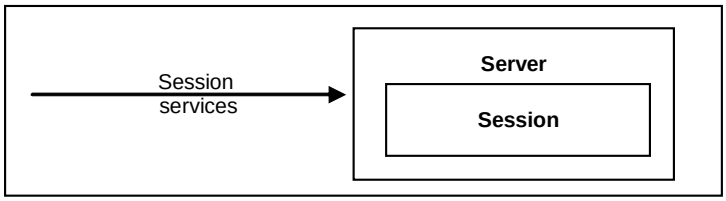
Légende

Anglais	Français
SecureChannel services	Services SecureChannel
Server	*Serveur*
Security Policy	Police de sécurité
Security Token	Jeton de sécurité

Figure 2 – Jeu de services de voie sécurisée (SecureChannel Service Set)

Le *Session Service Set* (Jeu de services de session), illustré à la Figure 3, définit les *Services* qui permettent au *Client* d'authentifier l'Utilisateur avec lequel il communique et de gérer des *Sessions*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

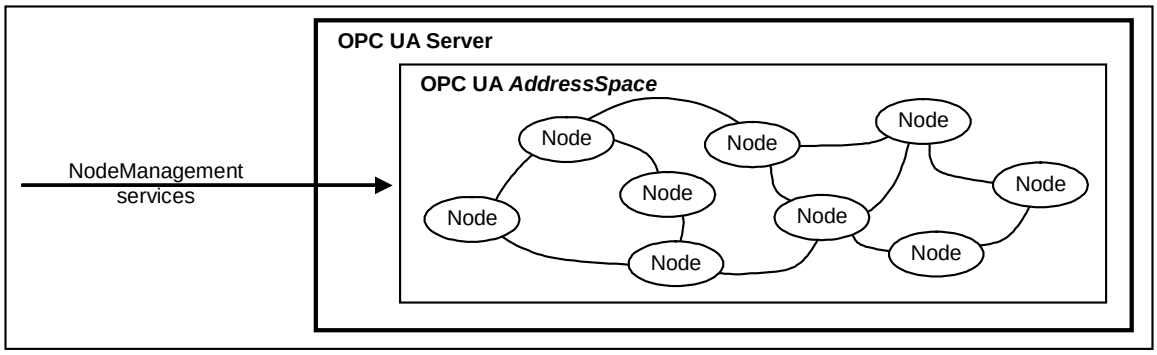


Légende

Anglais	Français
Session services	Services Session
Server	*Serveur*
Session	Session

Figure 3 – Jeu de services de session (Session Service Set)

Le *NodeManagement Service Set* (Jeu de services de gestion de nœuds), illustré à la Figure 4, définit les *Services* qui permettent au *Client* d'ajouter, de modifier et de supprimer des *Noeuds* (Nœuds) dans l'*AddressSpace* (Espace d'Adresses).



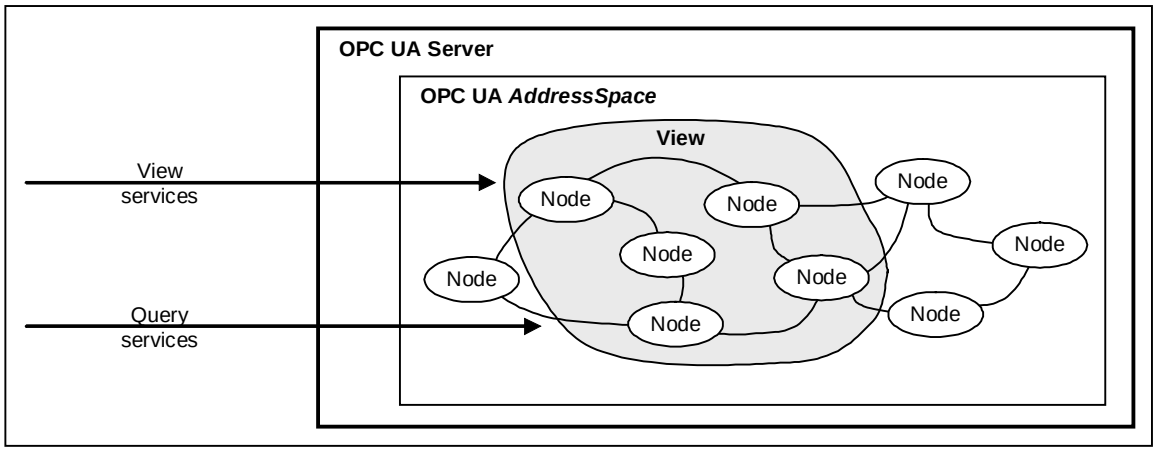
Légende

Anglais	Français
NodeManagement services	Services NodeManagement
OPC UA Server	*Serveur* OPC UA
OPC UA AddressSpace	Espace d'adresses OPC UA
Node	Nœud

Figure 4 – Jeu de services de gestion de nœuds (NodeManagement Service Set)

Le View Service Set (Jeu de services de vues), illustré à la Figure 5, définit les Services qui permettent aux Clients de parcourir l'Espace d'Adresses ou des sous-ensembles de l'Espace d'Adresses appelés Views (Vues). Le Query Service Set (Jeu de services d'interrogation) permet aux Clients de récupérer un sous-ensemble de données de l'Espace d'Adresses ou de la *Vue.*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



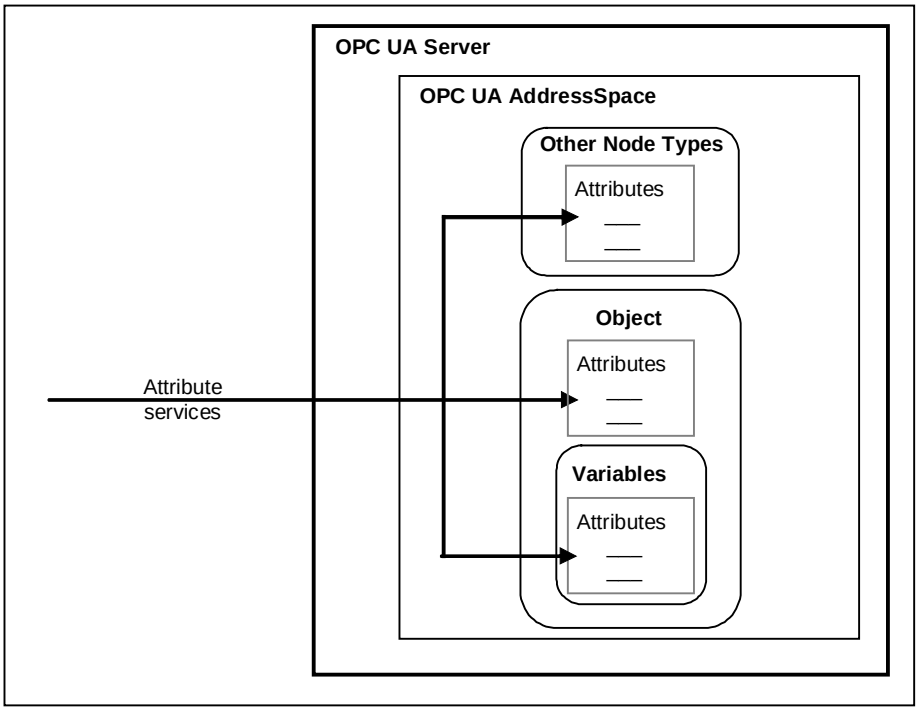
Légende

Anglais	Français
View services	Services View (vues)
Query services	Services Query (interrogation)
OPC UA Server	*Serveur* OPC UA
OPC UA AddressSpace	Espace d'adresses OPC UA
Node	Nœud
View	Vue

Figure 5 – Jeu de services de vues (View Service Set)

L'*Attribute Service Set* (Jeu de services d'attributs) est illustré à la Figure 6. Il définit les *Services* qui permettent aux *Clients* de lire et écrire les *Attributs* de *Noeuds*, y compris les valeurs de l'historique. La valeur d'une *Variable* étant modélisée comme *Attribut*, ces *Services* permettent aux *Clients* de lire et écrire les valeurs des *Variables*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



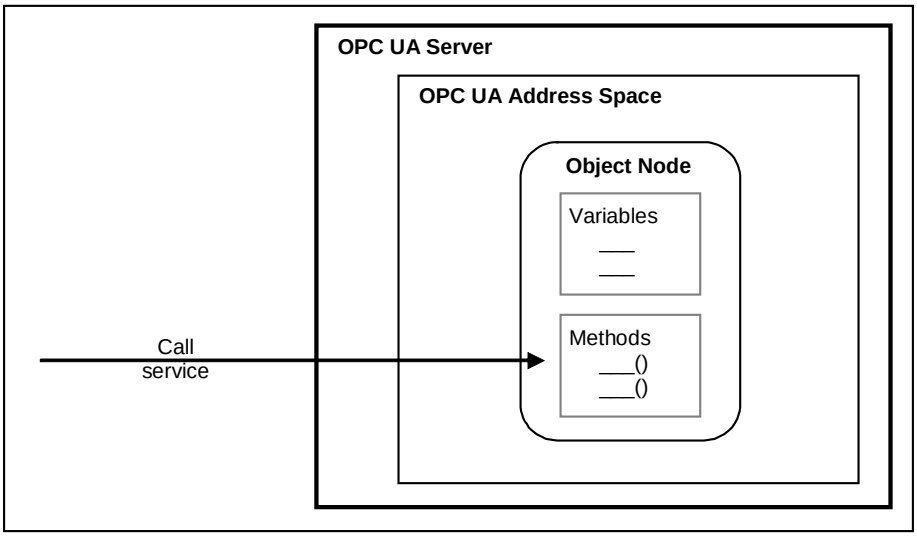
Légende

Anglais	Français
Attribute services	Services attributs
Other Node Types	Autres types de nœuds
OPC UA Server	*Serveur* OPC UA
OPC UA AddressSpace	Espace d'adresses OPC UA
Attributes	Attributs
Object	Objet
Variable	Variable

Figure 6 – Jeu de services d'attributs (Attribute Service Set)

Le *Method Service Set* (Jeu de services de méthodes) est illustré à la Figure 7. Il définit les *Services* qui permettent aux *Clients* d'appeler des méthodes. Les méthodes s'exécutent totalement lorsqu'elles sont appelées. Elles peuvent être appelées avec des paramètres d'entrée spécifiques à la méthode et peuvent retourner des paramètres de sortie spécifiques à la méthode.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



Légende

Anglais	Français
Call service	Service Appel
Object Node	Nœud objet
OPC UA Server	*Serveur* OPC UA
OPC UA AddressSpace	Espace d'adresses OPC UA
Methods	Méthodes
Variable	Variable

Figure 7 – Jeu de services de méthodes (Method Service Set)

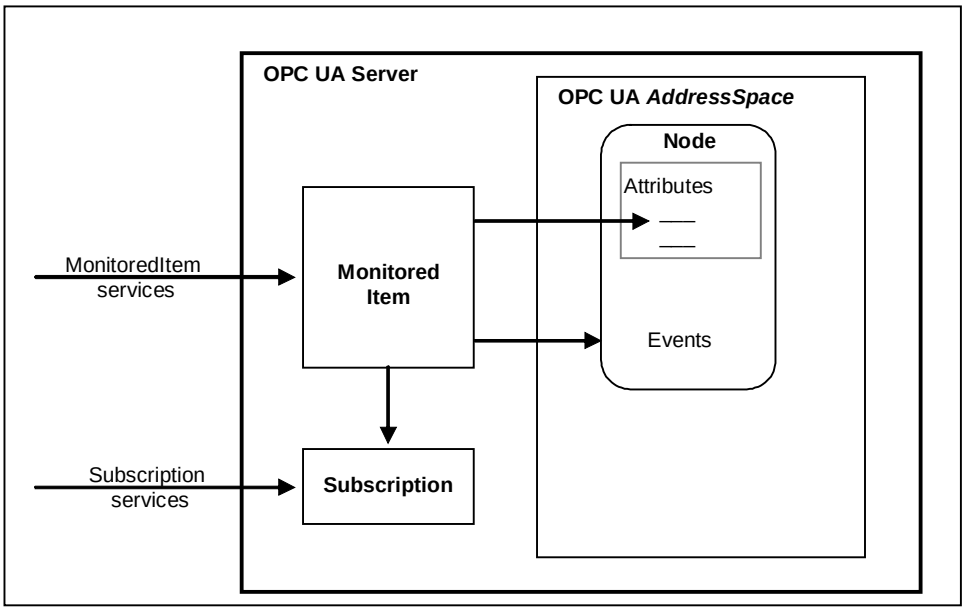
Le MonitoredItem Service Set (Jeu de services d'éléments surveillés) et le Subscription Service Set (Jeu de services d'abonnements), illustrés à la Figure 8, sont utilisés ensemble pour s'abonner à des *Noeuds* de *l'Espace d'Adresses* OPC UA.

Le *MonitoredItem Service Set* définit les *Services* qui permettent aux *Clients* de créer, modifier et supprimer des *MonitoredItems* (Eléments surveillés) utilisés pour surveiller les *Attributs* afin de déceler des changements de valeur et les *Objects* afin de détecter des *Evénements*.

Ces *Notifications* sont mises en file d'attente en vue de leur transfert vers le *Client* en utilisant le mécanisme de *Souscriptions* (Abonnements).

Le *Subscription Service Set* (Jeu de services d'abonnements) définit les *Services* qui permettent aux *Clients* de créer, modifier et supprimer des *Souscriptions*. Les *Souscriptions* envoient des *Notifications* générées par des *MonitoredItems* au *Client*. Les *Souscription Services* (Services d'abonnements) prévoient aussi la reprise *Client* après des *Messages* manqués et des défaillances de communications.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



Légende

Anglais	Français
MonitoredItem services	Services MonitoredItem (élément surveillé)
Subscription services	Services Abonnement
Node	Nœud
OPC UA Server	*Serveur* OPC UA
OPC UA AddressSpace	Espace d'adresses OPC UA
Attributes	Attributs
Events	Evénements
MonitoredItem	Elément surveillé
Subscription	Abonnement

Figure 8 – Jeux de services d'éléments surveillés et d'abonnements (MonitoredItem et Subscription Service Sets)

4.2 Procédures des services Demande/Réponses (Request/Response)

Les procédures des *Services* Request/Response décrivent le traitement des demandes et des réponses reçues par le *Serveur*. Les procédures débutent par la présentation au *Serveur* d'un *Message* de demande de *Service* de la part du *Client* demandeur.

A la réception de la demande, le *Serveur* traite le *Message* en deux étapes. Dans la première étape, il décode et localise le *Service* à exécuter. La gestion des erreurs pour cette étape est spécifique à la technologie de communication utilisée et elle est décrite dans la CEI 62541-6.

S'il réussit, il accède à chaque opération identifiée et exécute l'opération demandée. Pour chaque opération indiquée dans la demande, il génère un code de succès/échec distinct qu'il insère dans un *Message* de réponse positive accompagné des éventuelles données qui doivent être retournées.

Pour exécuter ces opérations, le *Client* et le *Serveur* peuvent se servir de l'API de la Pile Communication pour construire et interpréter les *Messages* et accéder à l'opération demandée.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

La mise en œuvre de la gestion de chaque demande de services ou réponse doit vérifier que chaque paramètre de service s'inscrit bien dans la plage spécifiée pour chaque paramètre.

5 Jeux de services (Service Sets)

5.1 Généralités

Le présent article définit les *Service Sets* d'OPC UA et leurs *Services*. L'Article 7 contient les définitions des paramètres communs utilisés par ces *Services*. Les exigences relatives à l'audit sont décrites en 6.2 pour tous les services.

Le *Server Profile* définit si un *Serve*ur prend ou non en charge un *Service Set*, ou un *Service* inclus dans un *Service Set*. Les *Profiles* sont décrits dans la CEI 62541-7.

5.2 En-tête de demande de services et de réponse

Chaque demande de *Service* à un *RequestHeader* (En-tête de demande) et chaque réponse relative à un *Service* a un *ResponseHeader* (En-tête de réponse).

La structure du *RequestHeader* est définie en 7.26 et contient les paramètres communs aux demandes tels qu'*authenticationToken* (Jeton d'authentification), *timestamp* (Marqueur temporel ou horodatage) et *requestHandle* (Gestion de demande).

La structure du *ResponseHeader* est définie en 7.27 et contient des paramètres communs aux réponses tels que *serviceResult* (Résultat de service) et *diagnosticInfo* (Informations de diagnostic).

5.3 Résultats des services

Les résultats d'un *Service* sont communiqués à deux niveaux dans les réponses OPC UA: l'un indique le statut de l'appel de *Service* tandis que l'autre indique le statut de chaque opération demandée par le *Service*.

Les résultats de *Service* sont définis via le *StatusCode* (Code de statut) (voir 7.33).

Le statut de l'appel de *Service* est représenté par le *serviceResult* contenu dans le *ResponseHeader* (voir 7.27). Le mécanisme pour retourner ce paramètre est spécifique à la technologie de communications utilisée pour acheminer la réponse de *Service* et il est défini dans la CEI 62541-6.

Le statut des opérations individuelles d'une demande est représenté par des *StatusCode*s individuels.

Les cas ci-après définissent l'utilisation de ces paramètres.

a) Un code bad (Mauvais) est retourné dans le *serviceResult* si le *Service* lui-même a failli. Dans ce cas, un *ServiceFault* (Défaut de service) est renvoyé. Le *ServiceFault* est défini en 7.28.

b) Le code good (Bon) est retourné dans le *serviceResult* si le *Service* a partiellement ou totalement réussi. Dans ce cas, d'autres paramètres de réponse sont retournés. Le *Client* doit toujours vérifier les paramètres de réponse, notamment les *StatusCodes* associés à chaque opération. Ces *StatusCodes* peuvent indiquer des résultats incorrects ou incertains pour une ou plusieurs opérations demandées lors de l'appel de *Service*.

Tous les *Services* avec des matrices d'opérations dans la demande doivent retourner un code bad dans le *serviceResult* si la matrice est vide.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les *Services* définissent divers *StatusCodes* spécifiques et un *Serveur* doit utiliser ces *StatusCodes* spécifiques comme décrit dans le *Service*. Il convient qu'un *Client* puisse gérer ces *StatusCodes* spécifiques au *Service*. En outre, un *Client* doit prendre en compte d'autres *StatusCodes* communs définis au Tableau 165 et au Tableau 166. Des détails complémentaires pour la gestion de *StatusCodes* spécifiques par le *Client* peuvent être définis dans la CEI 62541-7.

Si le *Serveur* découvre, par des mécanismes hors bande, que l'intégrité des authentifiants de l'application ou de l'utilisateur utilisés pour créer une *Session* ou un *Canal Sécurisé* a été compromise, le *Serveur* doit interrompre immédiatement toutes les sessions et fermer les voies qui utilisent ces authentifiants. Dans ce cas, le code de résultat du *Service* est soit *Bad_IdentityTokenRejected* ou *Bad_CertificateUntrusted*.

L'analyse de message peut échouer en raison d'erreurs de syntaxe ou si les données du message sortent de la plage prise en charge par le récipiendaire. Lorsque cela se produit, les messages doivent être rejetés par le récipiendaire. Si le récipiendaire est un *Server*, il doit retourner un *ServiceFault* avec le code de résultat *Bad_DecodingError* ou *Bad_EncodingLimitsExceeded*. Si le récepteur est le *Client*, il convient que la *Pile de Communication* rende compte de ces erreurs à l'application *Client*.

De nombreuses applications imposent des limites à la taille des messages et/ou des éléments de données contenus dans ces messages. Par exemple, un *Serveur* peut rejeter des demandes contenant des valeurs chaînes plus longues qu'une certaine longueur. Ces limites sont typiquement établies par des administrateurs et s'appliquent à toutes les connexions entre un *Client* et un *Serveur*.

Les *Clients* qui reçoivent des défauts *Bad_EncodingLimitsExceeded* de la part du *Serveur* auront vraisemblablement à reformuler leurs demandes. L'administrateur peut avoir besoin d'augmenter les limites pour le *Client* s'il reçoit une réponse du *Serveur* comportant ce défaut.

Dans certains cas, les erreurs d'analyse sont fatales et il n'est pas possible de retourner un défaut. Par exemple, le message entrant pourrait dépasser la capacité du tampon du récipiendaire. Dans ces cas, ces erreurs peuvent être traitées comme un défaut de communication qui exige le rétablissement de la *Canal Sécurisé* (voir 5.5).

Le *Client* et le *Serveur* réduisent les risques d'erreur fatale en échangeant leurs limites de taille de message dans le service *CreateSession*. Cela permet à chaque partie prenante d'éviter d'envoyer un message qui génère un défaut de communication. Il convient que le *Serveur* retourne un défaut *Bad_ResponseTooLarge* si un message de réponse dépasse la taille de message spécifiée par le *Client*. De même, il convient que la pile communication (*Communication Stack*) du *Client* rapporte une erreur *Bad_RequestTooLarge* à l'application avant d'envoyer un message qui dépasse la limite du *Serveur*.

A signaler que les limites de taille de message s'applique seulement au corps de message et n'inclut pas les en-têtes ou les effets d'une application d'une quelconque politique de sécurité. Cela signifie qu'un corps de message qui est plus petit que le maximum spécifié peut tout de même générer une erreur fatale.

5.4 Jeu de services découverte (Discovery Service Set)

5.4.1 Vue générale

Le *Service Set* définit les *Services* utilisés pour découvrir les *Endpoints* (*Points Terminaux)* implémentés par un *Serveur* et pour lire la configuration de sécurité pour ces *Endpoints*. Les *Discovery Services* (Services Découverte) sont mis en œuvre par des *Serveurs* individuels et par des *Discovery Servers* (Serveurs de découverte) spécialisés. La CEI 62541-12 décrit comment les *Discovery Servers* spécialisés utilisent les *Services Discovery*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Chaque *Serveur* doit avoir un *Discovery Endpoint* (Point d'extrémité de découverte) auquel les *Clients* accèdent sans établir une *Session*. Cet *Endpoint* peut ou peut ne pas être le même *Session Endpoint* (Point d'extrémité de session) que les *Clients* utilisent pour établir un *Canal Sécurisé.* Les Clients lisent les informations de sécurité nécessaires pour établir un *Canal Sécurisé* en appelant le *GetEndpoints Service* (Service de récupération des points d'extrémités) sur le *Discovery Endpoint*.

En outre, les *Serveurs* peuvent s'enregistrer auprès d'un *Discovery Server* connu en utilisant le service *RegisterServer* (Enregistrement de serveur). Les Clients peuvent découvrir plus tard tous les éventuels *Serveurs* enregistrés en appelant le *FindServers Service* (Service Trouver des serveurs) sur le *Discovery Server*.

Le processus de découverte complet est illustré à la Figure 9.

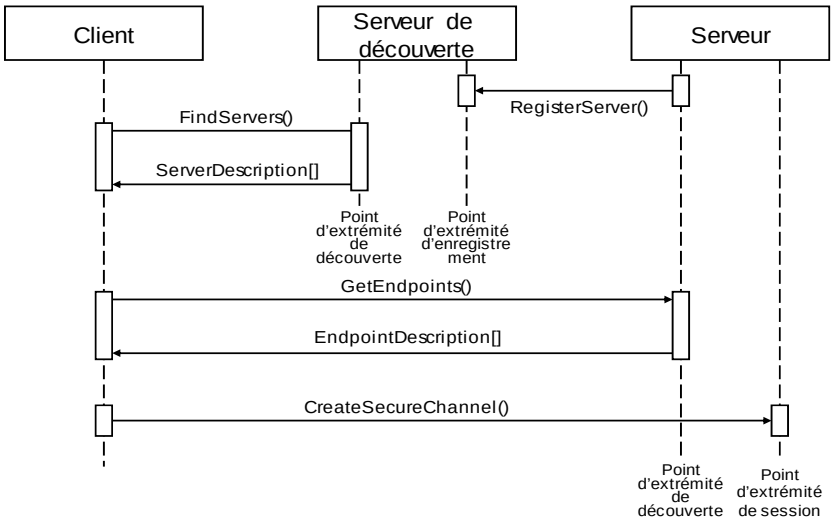


Figure 9 – Processus de découverte

L'URL (localisateur uniforme de ressource, *Uniform Resource Locator*) pour un *Discovery Endpoint* doit fournir toutes les informations dont le *Client* a besoin pour se connecter au *Discovery Endpoint*.

Une fois qu'un *Client* récupère les *Endpoints*, le *Client* peut sauvegarder cette information et l'utiliser pour se reconnecter directement au *Serveur* sans passer par le processus de découverte. Si le *Client* s'aperçoit qu'il ne peut pas se connecter, la configuration du *Serveur* a pu changer et il est nécessaire que le *Client* repasse par le processus de découverte.

Les *Discovery Endpoints* ne nécessitent aucune sécurité de message, mais elles peuvent requérir une sécurité de la couche transport. Dans les systèmes de production, les Administrateurs peuvent désactiver la découverte pour des raisons de sécurité et les *Clients* doivent compter sur des *EndpointDescriptions* (descriptions de points d'extrémité) placées en cache. Pour prendre en charge des systèmes ayant les *Discovery Services* désactivé, les *Clients* doivent permettre aux *Administrateurs* de mettre à jour manuellement les *EndpointDescriptions* utilisées pour se connecter à un *Serveur*. Les *Serveurs* doivent permettre aux *Administrateurs* de désactiver le *Discovery Endpoint*.

Le *Client* doit être très prudent pour utiliser les informations retournées par un *Discovery Endpoint* car elles ne sont pas sures. Le *Client* le fait en comparant les informations

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

retournées par le *Discovery Endpoint* aux informations retournées dans la réponse *CreateSession*. Le *Client* doit s'assurer que:

a) le *HostName* (Nom d'hôte) spécifié dans le *Serveur Certificate* est le même que le *HostName* contenu dans l'*endpointUrl* (URL de point d'extrémité) fourni dans l'*EndpointDescription*;

b) Le *Server Certificate* retourné dans la réponse *CreateSession* est le même que le *Certificate* utilisé pour créer le *Canal Sécurisé;*

c) Les *EndpointDescriptions* retournées par le *Discovery Endpoint* sont les mêmes que les *EndpointDescriptions* retournées dans la réponse *CreateSession*.

Si le *Client* détecte que l'une des exigences ci-dessus n'est pas remplie, le *Client* doit fermer le *Canal Sécurisé* et émettre un rapport d'erreur.

5.4.2 Trouver des serveurs (FindServers)

5.4.2.1 Description

Ce *Service* retourne les *Serveurs* connus auprès d'un *Serveur* ou d'un *Discovery Server*. Le comportement des *Discovery Servers* est décrit en détail dans la CEI 62541-12.

Le *Client* peut réduire le nombre de résultats retournés en spécifiant des critères de filtrage. Un *Discovery Server* retourne une liste vide si aucun *Serveur* ne correspond aux critères spécifiés par le client. Les critères de filtrage pris en charge par ce *Service* sont décrits en 5.4.2.2.

Chaque *Serveur* doit fournir un *Discovery Endpoint* qui prend en charge ce *Service*; cependant, le *Serveur* ne doit retourner qu'un seul enregistrement qui le décrit. Les serveurs passerelles *(Gateway Servers)* doivent retourner un enregistrement pour chaque *Serveur* auquel ils accordent l'accès et en plus de manière facultative un enregistrement qui permet au *Gateway Server* (serveur passerelle) d'être accessible comme un *Serveur* OPC UA ordinaire.

Chaque *Serveur* doit avoir un identificateur unique au niveau global appelé *serverUri*. Il convient que cet identificateur soit un nom de domaine totalement qualifié; cela peut être un GUID ou construct similaire qui assure l'unicité au niveau global. Le *serverUri* retourné par ce *Service* doit avoir la valeur qui apparaît en indice 0 de la propriété *ServerArray* (matrice de serveurs) (voir la CEI 62541-5).

Chaque *Serveur* doit aussi avoir un identificateur lisible par l'homme appelé *ServerName* (nom de serveur) qui au niveau global n'est pas nécessairement unique. Cet identificateur peut être disponible dans plusieurs paramètres de lieu.

Un *Serveur* peut avoir plusieurs *HostNames*. C'est pourquoi, le *Client* doit passer à ce *Service* l'URL qu'il a utilisé pour se connecter à l'*Endpoint*. L'implémentation de ce *Service* utilise cette information pour retourner des réponses qui sont accessibles au *Client* via l'URL fourni.

Ce *Service* ne nécessite aucune sécurité de message, mais il peut requérir une sécurité de la couche transport.

Certains *Serveurs* peuvent être accessibles via un *Gateway Server* et doivent avoir une valeur spécifiée pour *gatewayServerUri* (URI de serveur passerelle) dans leur *ApplicationDescription* (voir 7.1). Les *discoveryUrls* fournis dans *ApplicationDescription* doivent appartenir au *Gateway Server.* Certains *Discovery Servers* peuvent retourner plusieurs enregistrements pour le même *Serveur* si le *Serveur* en question peut être accessible par plusieurs chemins.

Ce *Service* peut être utilisé sans sécurité et il est donc vulnérable aux attaques par déni de service (DOS, *Denial Of Service*). Il convient qu'un *Serveur* réduise au maximum les

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

traitements requis pour envoyer une réponse via ce *Service*. Cela peut être fait en préparant le résultat à l'avance.

5.4.2.2 Paramètres

Le Tableau 3 définit les paramètres pour le *Service*.

Tableau 3 – Paramètres du service FindServers

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. L'*authenticationToken* (jeton d'authentification) est toujours omis. Le type *RequestHeader* est défini en 7.26.
endpointUrl	String	L'adresse réseau que le *Client* a utilisée pour accéder au *Discovery Endpoint*. Le *Serveur* utilise cette information pour des diagnostics et pour déterminer les URL à retourner dans la réponse. Il convient que le *Serveur* retourne un URL par défaut adéquat s'il ne reconnaît pas le *HostName* dans l'URL.
localeIds []	LocaleId	Liste des paramètres de lieu à utiliser. Il convient que le serveur retourne le *ServerName* en utilisant un des paramètres de lieu spécifiés. Si le serveur prend en charge plus d'un des paramètres de lieu demandés, il doit utiliser celui qui apparaît le premier dans la liste. Si le serveur ne prend en change aucun des paramètres de lieu demandés, il choisit un paramètre de lieu adéquat par défaut. Le serveur choisit un paramètre de lieu adéquat par défaut si cette liste est vide.
serverUris []	String	Liste des serveurs à retourner. Tous les serveurs connus sont retournés si la liste est vide.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses. Le type *ResponseHeader* est défini en 7.27.
servers []	ApplicationDescription	Liste des *Serveurs* qui répondent aux critères spécifiés dans la demande. Cette liste est vide si aucun serveur ne répond aux critères. Le type *ApplicationDescription* est défini en 7.1.

5.4.2.3 Résultats des services

Les *StatusCodes* communs sont définis au Tableau 165.

5.4.3 Déterminer les points d'extrémitée (GetEndpoints)

5.4.3.1 Description

Ce *Service* retourne les *Endpoints* pris en charge par un *Serveur* et la totalité des informations de configuration nécessaires pour établir un *Canal Sécurisé* et une *Session*.

Cela ne nécessite aucune sécurité pour les messages, mais peut requérir une sécurité de la couche transport.

Un *Client* peut réduire le nombre de résultats retournés en spécifiant des critères de filtrage. Le *Serveur* retourne une liste vide si aucun *Endpoint* ne correspond aux critères spécifiés par le client. Les critères de filtrage pris en charge par *Service* sont décrits en 5.4.3.2.

Un *Serveur* peut prendre en charge plusieurs configurations de sécurité pour le même *Endpoint*. Dans cette situation, le *Serveur* doit retourner des enregistrements d'*EndpointDescription* distincts pour chaque configuration disponible. Il faut que les *Clients* traitent chacune de ces configurations comme des *Endpoints* distincts même si l'URL physique se trouve être le même.

La configuration de sécurité pour un *Endpoint* comporte quatre composants:

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Server Application Instance Certificate (Certificat d'instance d'application serveur)

Message Security Mode (Mode de sécurité de message)

Security Policy (Politique de sécurité)

Supported User Identity Tokens (Jetons d'identité d'utilisateur pris en charge)

L'*ApplicationInstanceCertificate* est utilisé pour sécurisé la demande d'*OpenSecureChannel* (voir 5.5.2). Le *MessageSecurityMode* et le *SecurityPolicy* indiquent au *Client* comment sécuriser les messages envoyés via le *Canal Sécurisé*. Les *UserIdentityTokens* (jetons d'identité d'utilisateur) indiquent au client le type d'authentifiant d'utilisateur qui doit être passé au *Serveur* dans la demande *ActivateSession* (voir 5.6.3).

Chaque *EndpointDescription* spécifie aussi un URI pour le *profil de transport* que l'*Endpoint* prend en charge. Les *Transport Profiles* (Profils de transport) spécifient les informations telles que le format de codage de message et la version de protocole et sont définis dans la CEI 62541-7. Les *Clients* doivent chercher les *SoftwareCertificates* (Certificats de logiciel) du *Serveur* s'ils veulent découvrir la liste complète des *Profiles* pris en compte par le *Serveur* (voir 7.30).

Les messages sont sécurisés en appliquant aux messages avant de les envoyer sur le réseau des algorithmes de chiffrement normalisés. Le jeu exact d'algorithmes utilisés dépend de la *SecurityPolicy* pour l'*Endpoint*. La CEI 62541-7 définit des *Profiles* pour des *SecurityPolicies* communes et leur affecte un URI unique. Il est prévu que les applications connaissent les *SecurityPolicies* (Politiques de sécurité) qu'elles prennent en charge et, par conséquent, seul l'URI de profil pour la *SecurityPolicy* est spécifiée dans l'*EndpointDescription*. Un *Client* ne peut pas se connecter à un *Endpoint* qui ne prend pas en charge une *SecurityPolicy* qu'il connaît.

Un *EndpointDescription* peut spécifier que le mode de sécurité de message est NONE (Aucun/Rien). Cette configuration n'est pas recommandée sauf si les applications communiquent sur un réseau physiquement isolé dans lequel le risque d'intrusion est extrêmement faible. Si la sécurité de message est NONE, il est possible pour le *Clients* de pirater, délibérément ou accidentellement, des *Sessions* créées par d'autres *Clients*.

Un *Serveur* peut avoir plusieurs *HostNames*. Pour cette raison, le *Client* doit passer à ce *Service* l'URL qu'il a utilisé pour se connecter à l'*Endpoint*. L'implémentation de ce *Service* doit utiliser cette information pour retourner des réponses qui sont accessibles au *Client* via l'URL fourni.

Ce *Service* peut être utilisé sans sécurité et il est donc vulnérable aux attaques par déni de service (DOS, Denial Of Service). Il convient qu'un *Serveur* réduise au maximum la quantité de traitement nécessaire pour envoyer la réponse à ce *Service*. Cela peut être fait en préparant le résultat à l'avance.

Certaines des *EndpointDescriptions* retournées dans une réponse doivent spécifier les informations d'*Endpoint* pour un *Gateway Server* qui peut être utilisé pour accéder à un autre *Serveur*. Dans ce cas, le *gatewayServerUri* (l'URI de serveur passerelle) est spécifié dans l'*EndpointDescription* et tous les contrôles de sécurité utilisés pour vérifier les *Certificates* doivent utiliser le *gatewayServerUri* (voir 6.1.4) au lieu du *serverUri* (URI du serveur).

Pour se connecter à un *Serveur* via la passerelle, le *Client* doit d'abord établir un *Canal Sécurisé* avec le *Gateway Server*. Le *Client* doit appeler le service *CreateSession* et passe au *Gateway Server* le *serverUri* spécifié dans l'*EndpointDescription*. Le *Gateway Server* doit alors se connecter au *Serveur* sous-jacent pour le compte du *Client*. Le processus de connexion au *Serveur* via un *Gateway Server* est illustré à la Figure 10.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

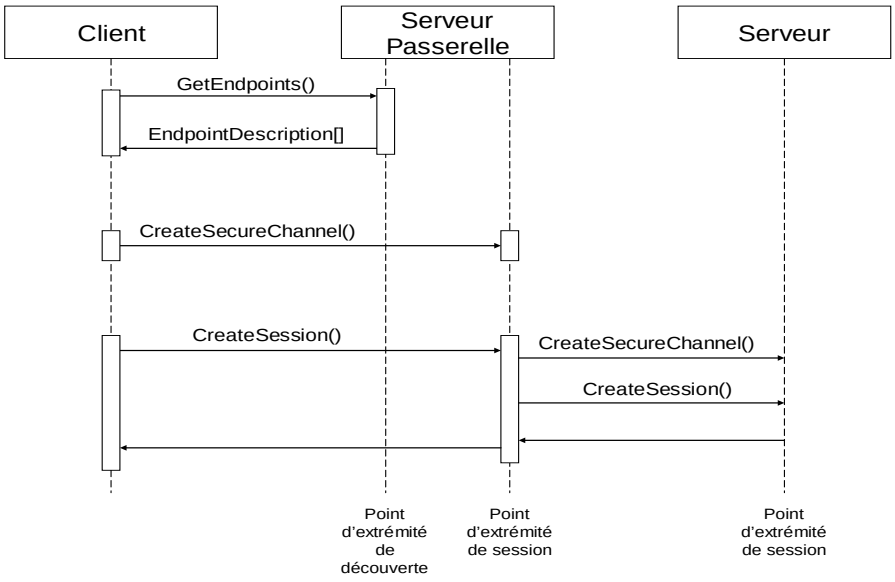


Figure 10 – Utilisation d'un Gateway Server (Serveur passerelle)

5.4.3.2 Paramètres

Le Tableau 4 définit les paramètres pour le *Service*.

Tableau 4 – Paramètres du service GetEndpoints

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. L'*authenticationToken* (jeton d'authentification) est toujours omis. Le type *RequestHeader* est défini en 7.26.
endpointUrl	String	L'adresse réseau que le *Client* a utilisée pour accéder au *Discovery Endpoint*. Le *Serveur* utilise cette information pour des diagnostics et pour déterminer les URL à retourner dans la réponse. Il convient que le *Serveur* retourne un URL adéquat par défaut s'il ne reconnaît pas le *HostName* dans l'URL.
localeIds []	LocaleId	Liste des paramètres de lieu à utiliser. Spécifie le paramètre de lieu à utiliser pour retourner des chaînes lisibles par l'homme. Ce paramètre est décrit en 5.4.4.2.
profileUris []	String	Liste de profils que doivent prendre en charge les *Endpoints* retournés. Tous les *Endpoints* sont retournés si la liste est vide.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses. Le type *ResponseHeader* est défini en 7.27.
Endpoints []	EndpointDescription	Liste des *Endpoints* qui répondent aux critères spécifiés dans la demande. Cette liste est vide si aucun *Endpoint* ne répond aux critères. Le type *EndpointDescription* est défini en 7.9.

5.4.3.3 Résultats des services

Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.4.4 Enregistrer un serveur (RegisterServer)

5.4.4.1 Description

Ce *Service* enregistre un *Serveur* auprès d'un *Discovery Server* (Serveur de découverte). Ce *Service* sera appelé par un *Serveur* ou par un utilitaire de configuration distinct. Les *Clients* n'utiliseront pas ce *Service*.

Un *Serveur* doit établir un *Canal Sécurisé* avec le *Discovery Server* avant d'appeler ce *Service*. Le *Canal Sécurisé* est décrit en 5.5. L'*Administrateur* du *Serveur* doit fournir au *Serveur* une *EndpointDescription* pour le *Discovery Server, cela fait* partie intégrante du processus de configuration. Les *Discovery Servers* doivent rejeter les enregistrements si le *serverUri* fourni ne correspond pas à l'*applicationUri* contenue dans le *Server Certificate* utilisé pour créer le *Canal Sécurisé*.

Un *Serveur* fournit uniquement son *serverUri* et les URL des *Discovery Endpoints* au *Discovery Server*. Les *Clients* doivent utiliser le service *GetEndpoints* pour rechercher directement auprès du *Serveur* les informations de configurations les plus récentes.

Le *Serveur* doit fournir un nom localisé pour lui-même dans tous paramètres de lieu qu'il prend en charge.

Les *Serveurs* doivent être à même de s'enregistrer auprès d'un *Discovery Server* tournant sur la même machine. Les mécanismes exacts dépendent de l'implémentation du *Discovery Server* et sont décrits dans la CEI 62541-6.

Il existe deux types d'applications d'un *Serveur*: celles qui sont lancées manuellement et celles qui sont lancées automatiquement lorsqu'un *Client* tente de se connecter. Le processus d'enregistrement qu'un *Serveur* doit utiliser dépend de la catégorie dans laquelle il s'inscrit.

Le processus d'enregistrement pour les *Serveurs* lancés manuellement est illustré à la Figure 11.

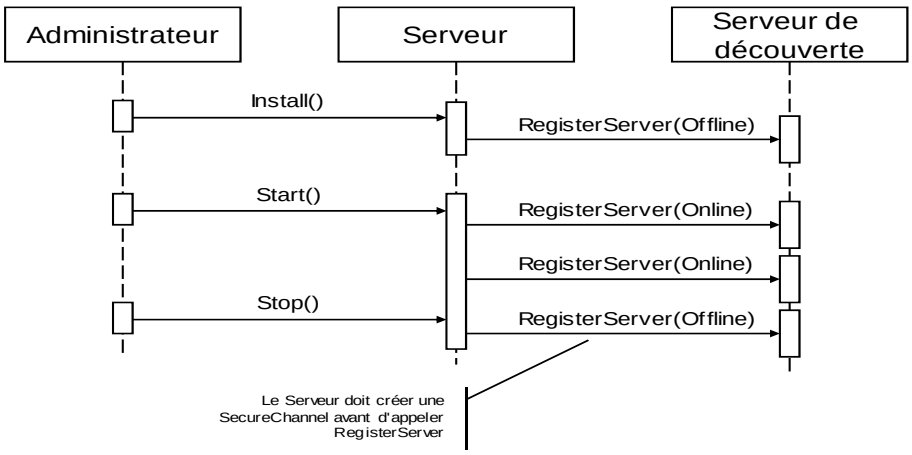


Figure 11 – Processus d'enregistrement – Serveurs lancés manuellement

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le processus d'enregistrement pour les *Serveurs* lancés automatiquement est illustré à la Figure 12.

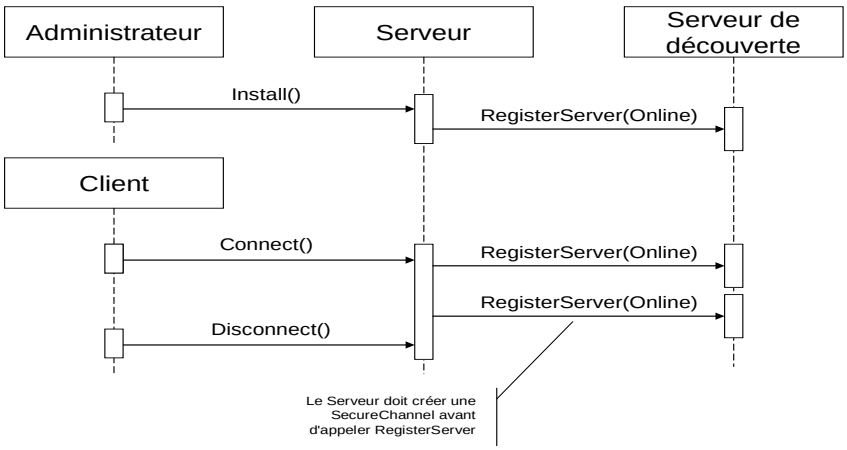


Figure 12 – Processus d'enregistrement – Serveurs lancés automatiquement

Le processus d'enregistrement est conçu pour être indépendant de la plateforme, être robuste et capable de réduire au maximum les erreurs créées par une mauvaise configuration. Pour cette raison, les *Serveurs* doivent s'enregistrer plus d'une fois.

Dans des conditions normales, les *Serveurs* doivent s'enregistrer régulièrement auprès du *Discovery Server* tant qu'ils peuvent recevoir des connexions des *Clients*. Si un *Serveur* passe hors ligne, il doit s'enregistrer une fois de plus et indiquer s'il passe hors ligne. Il convient que la fréquence d'enregistrement soit configurable; cependant, la valeur par défaut est de 10 min.

Si une erreur se produit au cours de l'enregistrement (par exemple le *Discovery Server* n'est pas en marche), le *Serveur* doit de nouveau tenter périodiquement de s'enregistrer. Il convient que la fréquence de ces tentatives commence à 1 s mais augmente progressivement jusqu'à ce que la fréquence d'enregistrement atteigne la valeur qu'elle aurait eu si aucune erreur ne s'était produite. L'approche recommandée double la période de chaque tentative jusqu'à ce que le maximum soit atteint.

Lorsqu'un *Serveur* s'enregistre auprès d'un *Discovery Server*, il peut choisir de fournir un fichier sémaphore que le *Discovery Server* peut utiliser pour déterminer si le *Serveur* a été désinstallé de la machine. Le *Discovery Server* doit avoir eu accès au système de fichiers qui contient le fichier.

5.4.4.2 Paramètres

Le Tableau 5 définit les paramètres pour le *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 5 – Paramètres du service RegisterServer

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. L'*authenticationToken* (jeton d'authentification) est toujours omis. Le type *RequestHeader* est défini en 7.26.
Server	RegisteredServer	Le serveur à enregistrer.
serverUri	String	L'identificateur unique au niveau global pour l'instance de *Serveur*.
productUri	String	L'identificateur unique au niveau global pour le produit *Serveur*.
serverNames []	LocalizedText	Une liste des noms descriptifs localisés pour le *Serveur*. La liste doit avoir au moins une entrée valide.
serverType	Enum ApplicationType	Le type d'application. Les valeurs d'énumération sont définies au Tableau 103. La valeur “CLIENT_1” (L'application est un *Client*.) n'est pas autorisée.
gatewayServerUri	String	L'URI du *Gateway Server* associé aux *discoveryUrls*. Cette valeur est uniquement spécifiée par les *Gateway Servers* qui souhaitent enregistrer les *Serveurs* auxquels ils donnent accès. Pour les *Serveurs* qui n'agissent pas comme un *Gateway Server*, ce paramètre doit être null.
discoveryUrls []	String	Une liste de *Discovery Endpoints* pour le *Serveur*. La liste doit avoir au moins une entrée valide.
semaphoreFilePath	String	Le chemin menant au fichier sémaphore utilisé pour identifier l'instance de serveur. Le *Discovery Server* doit s'assurer que ce fichier existe avant de retourner l'*ApplicationDescription* au client. Si le même fichier sémaphore est utilisé par un autre *Serveur*, cet enregistrement est supprimé et remplacé par celui actuellement transmis comme partie de l'invocation de service. Si la valeur est null ou vide, le *DiscoveryServer* ne tente pas de vérifier l'existence du fichier.
isOnline	Boolean	True (vrai) si le *Serveur* est actuellement capable d'accepter des connexions de Clients.
Response		
ResponseHeader	ResponseHeader	Paramètres communs aux réponses. Le type *ResponseHeader* est défini en 7.27.

5.4.4.3 Résultats des services

Le Tableau 6 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 6 – Codes des résultats de services RegisterServer

Id symbolique	Description
Bad_ServerUriInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ServerNameMissing	Aucun *ServerName* n'avait été spécifié.
Bad_DiscoveryUrlMissing	Aucun *DiscoveryUrl* n'avait été spécifié.
Bad_SempahoreFileMissing	Le fichier sémaphore spécifié n'est pas valide.

5.5 Jeu de services de voie sécurisée (SecureChannel Service Set)

5.5.1 Vue d'ensemble

Ce *Service Set* (jeu de services) définit les *Services* utilisés pour ouvrir une voie de communication qui assure la confidentialité et l'*Integrity* de tous les *Messages* échangés avec le *Serveur*. Les concepts de base pour la sécurité OPC UA sont définis dans la CEI/TR 62541-2.

Les *SecureChannel Services* (services de voie sécurisée) ne sont pas comme d'autres *Services* car ils ne sont pas implémentés directement par l'*Application OPC UA*. En revanche, ils sont fournis par la *Communication Stack* sur laquelle est bâtie l'*Application OPC UA*. Par exemple, un *Serveur* OPC UA peut être bâti sur une pile SOAP (Simple Object Access Protocol, Protocole d'accès à objet simple) qui permet à des applications d'établir un *Canal Sécurisé* en utilisant la spécification WS Secure Conversation. Dans ces cas, l'*Application*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

OPC UA doit vérifier que le *Message* qu'elle a reçu était dans le contexte d'une WS Secure Conversation. La CEI 62541-6 décrit comment les *Services SecureChannel* sont mises en œuvre.

Un *Canal Sécurisé* est une connexion logique établie entre un seul *Client* et un seul *Serveur*. Cette voie maintient un jeu de clés connues seulement du *Client* et du *Serveur*, qui sont utilisées pour authentifier et chiffrer des *Messages* à travers le réseau. Les *Services SecureChannel* permettent au *Client* et au *Serveur* de négocier en toute sécurité les clés à utiliser.

Une *EndpointDescription* indique au Client comment établir un *Canal Sécurisé* avec un *Endpoint* donné. Un *Client* peut obtenir l'*EndpointDescription* provenant d'un *Discovery Server*, via quelque serveur non défini par l'UA, ou provenant de sa propre configuration.

Les algorithmes exacts utilisés pour authentifier et chiffrer les *Messages* sont décrits dans le champ *SecurityPolicy* de l'*EndpointDescription*. Un *Client* peut utiliser ces algorithmes lorsqu'il crée un *Canal Sécurisé* .

A remarquer que certaines *SecurityPolicies* définies dans la CEI 62541-7 désactiveront l'authentification et le chiffrement, conduisant à un *Canal Sécurisé* qui ne procure aucune sécurité.

Lorsqu'un *Client* et un *Serveur* sont en communication via un *Canal Sécurisé*, ils doivent vérifier que tous les *Messages* entrants ont été signés et chiffrés conformément aux exigences spécifiées dans l'*EndpointDescription*. Une *Application OPC UA* ne doit traiter aucun *Message* qui n'est pas conforme à ces exigences.

La relation entre le *Canal Sécurisé* et l'*Application OPC UA* dépend de la technologie d'implémentation. La CEI 62541-6 définit toutes les exigences induitent par la technologie utilisée.

La corrélation entre une *Session* d'*Application OPC UA* et le *Canal Sécurisé* est illustrée à la Figure 13. La *Pile de Communication (Communication Stack)* est utilisée par les *Applications OPC UA* pour échanger des *Messages*. Dans une première étape, les *Services SecureChannel* sont utilisés pour établir un *Canal Sécurisé* entre les deux *Communication Stacks* qui permet l'échange sécurisé de *Messages*. Dans une seconde étape, les *Applications OPC UA* utilisent le *Session Service Set* (jeu de services de session) pour établir une *Session* d'*Application OPC UA*.

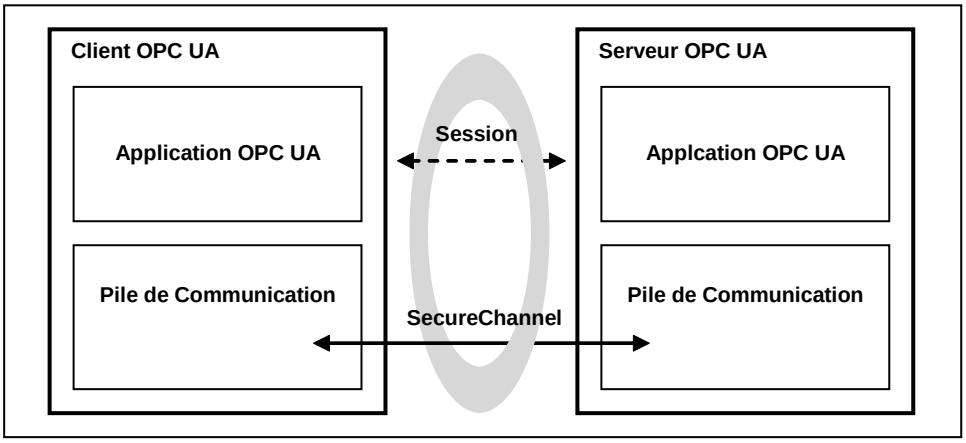


Figure 13 – Services SecureChannel et Session

Une fois qu'un *Client* a établi une *Session*, il peut souhaiter accéder à cette *Session* à partir d'un *Canal Sécurisé* différent. Le client peut le faire en validant le nouveau *Canal Sécurisé* avec le *Service ActivateSession* décrit en 5.6.3.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Si un *Serveur* agit comme *Client* auprès d'autres *Serveurs*, ce qui est communément appelé chaînage de *Serveurs*, le *Serveur* doit être à même de maintenir le niveau de sécurité utilisateur. Cela signifie qu'il convient que l'identité de l'utilisateur soit transmise au serveur sous-jacent ou qu'elle soit mise en correspondance avec une identité d'utilisateur appropriée dans le serveur sous-jacent. Il est inacceptable d'ignorer la sécurité de niveau utilisateur. Elle est indispensable pour s'assurer que la sécurité est maintenue et qu'un utilisateur n'obtiendra pas d'informations auxquelles il convient qu'il n'ait pas accès. Autant que possible, il convient qu'un *Serveur* se substitue au *Client* d'origine en transmettant l'identité d'utilisateur du *Client* d'origine au *Serveur* sous-jacent lorsqu'il appelle le *Service ActiveSession*. Si la substitution d'identité n'est pas une option, le *Serveur* doit faire correspondre l'identité d'utilisateur du *Client* avec une nouvelle identité d'utilisateur que le *Serveur* sous-jacent reconnaît effectivement.

5.5.2 Ouvrir un canal sécurisé (OpenSecureChannel)

5.5.2.1 Description

Ce *Service* est utilisé pour ouvrir ou renouveler un *Canal Sécurisé* qui peut être utilisée pour assurer la *Confidentiality* et l'*Integrity* pour l'échange de *Messages* pendant cette *Session*. Ce *Service* exige de la *Pile de Communication* qu'elle applique les divers algorithmes de sécurité aux *Messages* à mesure qu'ils sont envoyés et reçus. Des mises en œuvre spécifiques de ce *Service* pour différentes *Piles de Communication* (*Communication Stacks*) sont décrites dans la CEI 62541-6.

Chaque *Canal Sécurisé* a un identificateur unique au niveau global et est valide pour une combinaison spécifique d'instances d'applications *Client* et *Serveur*. Chaque voie contient un ou plusieurs *SecurityTokens* (jetons de sécurité) qui identifie un jeu de clés cryptographiques qui sont utilisées pour chiffrer et authentifier des *Messages*. Les *SecurityTokens* aussi ont des identificateurs uniques au niveau global qui sont joints à chaque *Message* sécurisé avec le jeton. Cela permet à un récipiendaire autorisé de savoir comment déchiffrer et vérifier les *Message*.

Les *SecurityTokens* ont une durée de vie finie négociée avec ce *Service*. Cependant, les différences entre les horloges système de plusieurs machines et les temps de latence réseau impliquent que des *Messages* valides pourraient arriver après l'expiration du jeton. Pour éviter que des *Messages* valides soient rejetés, les applications doivent exécuteer les actions suivantes:

a) il convient que les *Clients* demandent un nouveau *SecurityToken* après 75 % de leur durée de vie. De manière à ce que les *Clients* recoivent le nouveau *SecurityToken* avant que l'ancien n'expire effectivement;

b) il convient que les *Serveurs* utilisent le *SecurityToken* existant pour sécuriser les *Messages* sortant jusqu'à ce que le *SecurityToken* expire ou que le *Server* reçoive un *Message* sécurisé avec un nouveau *SecurityToken*. De manière à ce que les *Clients* ne rejettent pas des *Messages* sécurisés reçus avec le nouveau *SecurityToken* avant que le *Client* ne reçoive le nouveau *SecurityToken*;

c) il convient que les *Clients* acceptent des *Messages* sécurisés par un *SecurityToken* expiré pendant une durée pouvant atteindre 25 % de la durée de vie du jeton. De manière à ce que les *Messages* envoyés par le *Serveur* avant l'expiration du jeton ne soient pas rejetés en raison des retards du réseau.

Chaque *Canal Sécurisé* existe jusqu'à ce qu'il soit explicitement fermé ou jusqu'à ce que le dernier jeton ait expiré et que la période de chevauchement ait expiré.

Les *Messages* de demande d'*OpenSecureChannel* et de réponse doivent être signés avec le *Certificate* de l'expéditeur. Ces *Messages* doivent toujours être chiffrés. Si la couche transport ne fournit pas de chiffrement, ces *Messages* doivent être chiffrés avec le *Certificat* du récipiendaire.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les Certificats utilisés dans le service *OpenSecureChannel* doivent être les certificats d'instance d'application. *Clients* et *Serveurs* doivent vérifier que les mêmes Certificats ont été utilisés dans les services *CreateSession* et *ActivateSession*.

5.5.2.2 Paramètres

Le Tableau 7 définit les paramètres pour le *Service*.

Tableau 7 – Paramètres du service OpenSecureChannel

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. L'*authenticationToken* (jeton d'authentification) est toujours omis. Le type *RequestHeader* est défini en 7.26 .
clientCertificate	ApplicationInstance Certificate	Un *Certificat* qui identifie le *Client*. La demande d'*OpenSecureChannel* doit être signée avec ce *Certificat*. Le type *ApplicationInstanceCertificate* est défini en 7.2.
requestType	enum SecurityToken RequestType	Le type de demande de *SecurityToken*: Une énumération qui doit être l'une des suivantes: ISSUE_0 crée un nouveau *SecurityToken* pour un nouvel *Canal Sécurisé* . RENEW_1 crée un nouveau *SecurityToken* pour un *Canal Sécurisé* existant.
secureChannelId	ByteString	L'identificateur du *Canal Sécurisé* à laquelle il convient que le nouveau jeton appartienne. Ce paramètre doit être null lors de la création d'un nouveau *Canal Sécurisé* .
securityMode	Enum MessageSecurityMode	Le type de sécurité à appliquer aux messages. Le type *MessageSecurityMode* est défini en 7.14. Il se peut qu'un *Canal Sécurisé* doit être créée même si le *securityMode* est NONE. Le comportement exact dépend de la mise en correspondance utilisée et est décrit dans la CEI 62541-6.
securityPolicyUri	String	L'URI pour la *SecurityPolicy* à utiliser lors de la sécurisation des messages envoyés sur le *Canal Sécurisé* . Le jeu d'URI connus et les *SecurityPolicies* qui leur sont associées sont définis dans la CEI 62541-7.
clientNonce	ByteString	Un nombre aléatoire qui ne doit être utilisé par aucune autre demande. Un nouveau *clientNonce* doit être généré chaque fois qu'un *Canal Sécurisé* est renouvelé. Ce paramètre doit avoir une longueur égale à la taille de clé utilisée pour l'algorithme de chiffrement symétrique qui est identifié par le *securityPolicyUri*.
requestedLifetime	Duration	La durée de vie demandée, en millisecondes, pour le nouveau *SecurityToken*. Il spécifie quand le *Client* pense renouveler le *Canal Sécurisé* en appelant de nouveau le *Service OpenSecureChannel*. Si un *Canal Sécurisé* n'est pas renouvelée, tous les *Messages* envoyés en utilisant les *SecurityTokens* courants doivent être rejetés par le récipiendaire.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition du type *ResponseHeader*).
securityToken	ChannelSecurityToken	Décrit le nouveau *SecurityToken* émis par le *Serveur*.
ChannelID	ByteString	Un identificateur unique pour le *Canal Sécurisé* . Il s'agit de l'identificateur qui doit être fourni chaque fois que le *Canal Sécurisé* est renouvelé.
tokenId	ByteString	Un identificateur unique pour un seul *SecurityToken* dans la voie. Il s'agit de l'identificateur qui doit être transmis avec chaque *Message* sécurisé avec le *SecurityToken*.
createdAt	UTCTime	L'instant où le *SecurityToken* a été créé.
revisedLifetime	Duration	La durée de vie du *SecurityToken* en millisecondes. L'heure d'expiration TUC pour le jeton peut être calculée en ajoutant la durée de vie au temps *createdAt*.
serverNonce	ByteString	Un nombre aléatoire qui ne doit être utilisé par aucune autre demande. Un nouveau *serverNonce* doit être généré chaque fois qu'un *Canal Sécurisé* est renouvelé. Ce paramètre doit avoir une longueur égale à la taille de clé utilisée pour l'algorithme de chiffrement symétrique qui est identifié par le *securityPolicyUri*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.5.2.3 Résultats des services

Le Tableau 8 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 8 – Codes des résultats de services OpenSecureChannel

Id symbolique	Description
Bad_SecurityChecksFailed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateTimeInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerTimeInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateHostNameInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUriInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUseNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerUseNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUntrusted	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateRevocationUnknown	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerRevocationUnknown	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateRevoked	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerRevoked	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_RequestTypeInvalid	Le type de la demande de jeton de sécurité n'est pas valide.
Bad_SecurityModeRejected	Le mode de sécurité ne satisfait pas aux exigences établies par le *Serveur*.
Bad_SecurityPolicyRejected	La politique de sécurité ne satisfait pas aux exigences établies par le *Serveur*.
Bad_SecureChannelIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NonceInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.5.3 Fermer un canal sécurisé (CloseSecureChannel)

5.5.3.1 Description

Ce *Service* est utilisé pour mettre fin à un *Canal Sécurisé*.

Les *Messages* de demande doivent être signés avec la clé appropriée associée au jeton courant pour le *Canal Sécurisé* .

5.5.3.2 Paramètres

Le Tableau 9 définit les paramètres pour le *Service*.

Tableau 9 – Paramètres du service CloseSecureChannel

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. Le *sessionId* (identificateur de session) est toujours omis. Le type *RequestHeader* est défini en 7.26.
secureChannelId	ByteString	L'identificateur pour le *Canal Sécurisé* à fermer.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).

5.5.3.3 Résultats des services

Le Tableau 10 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 10 – Codes des résultats de services CloseSecureChannel

Id symbolique	Description
Bad_SecureChannelIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.6 Jeu de services de session (Session Service Set)

5.6.1 Vue générale

Ce *Service Set* définit des *Services* pour l'établissement de connexions de couche application dans le contexte d'une *Session*.

5.6.2 Créer une session (CreateSession)

5.6.2.1 Description

Ce *Service* est utilisé par un *Client* OPC UA pour créer une *Session* et le *Serveur* retourne deux valeurs qui identifient de façon unique cette *Session*. La première valeur est le *sessionId* qui est utilisé pour identifier cette *Session* dans les journaux d'audit dans l'Espace d'Adresses du *Serveur*. La seconde est l'*authenticationToken* qui est utilisé pour associer une demande entrante à une *Session*.

Avant d'appeler ce *Service*, le *Client* doit créer un *Canal Sécurisé* avec le *Service OpenSecureChannel* pour garantir l'*Integrity* de tous les *Messages* échangés au cours de cette *Session*. Ce *Canal Sécurisé* a un identificateur unique, que le *Serveur* doit associer à l'*authenticationToken*. Le *Serveur* peut accepter des demandes avec l'*authenticationToken* seulement si elles sont associées au*Canal Sécurisé* qui avait été utilisée pour créer cette *Session*. Le *Client* peut associer un nouvel *Canal Sécurisé* à la *Session* en appelant la méthode *ActivateSession*.

Le *Canal Sécurisé* est toujours géré par la *Pile de Communication*, ce qui signifie qu'il doit fournir les API que le *Serveur* peut utiliser pour trouver des informations du *Canal Sécurisé* utilisé pour n'importe quelle demande donnée. La *Communication Stack* doit, au minimum, fournir la *SecurityPolicy* et le *SecurityMode* utilisés par le *Canal Sécurisé* . Elle doit aussi fournir un *Canal Sécurisé Id* (identificateur de voie sécurisée) qui identifie de façon unique le *Canal Sécurisé* ou le *Client Certificate* (certificat de client) utilisé pour établir le *Canal Sécurisé*. Le *Serveur* utilise l'un de ceux-ci pour identifier le *Canal Sécurisé* utilisé pour envoyer une demande. Le paragraphe 7.29 décrit comment créer l'*authenticationToken* pour différents types de *Communication Stack*.

En fonction de la *SecurityPolicy* et du *SecurityMode* du *Canal Sécurisé,* l'échange des *Application Instance Certificates* (Certificats d'instances d'application) et des *Nonces (*number used once, nombre utilisé une seule fois*)* peut être facultatif et les signatures peuvent être vides. Voir la CEI 62541-7 pour la définition des *SecurityPolicies* et la gestion de ces paramètres.

Le *Serveur* retourne ses *EndpointDescriptions* dans la réponse. Les *Clients* utilisent cette information pour déterminer si, oui ou non, la liste des *EndpointDescriptions* retournées par le *Discovery Endpoint* correspond aux *Endpoints* que le *Serveur* possède. S'il y a une différence, le *Client* doit fermer cette *Session* et rapporter l'erreur. Le *Serveur* retourne toutes les *EndpointDescriptions* pour le *serverUri* spécifié par le *Client* dans la demande. Le *Client* vérifie uniquement les *EndpointDescriptions* ayant un *transportProfileUri* (URI de profil de transport) qui correspond au *profileUri* (URI de profil) spécifié dans la demande de *GetEndpoints* initiale. Un *Client* peut sauter cette vérification si les *EndpointDescriptions* ont été fournies par une source de confiance telle que l'*Administrateur*.

Une *Session* créée avec ce *Service* ne doit pas être utilisée jusqu'à ce que le *Client* appelle le *Service ActivateSession*, fournisse ses *SoftwareCertificates* et prouve la possession de son *Certificat* d'instance d'application et de tout jeton d'identité utilisateur qu'il fournit.

La réponse contient également une liste des *SoftwareCertificates* qui identifient les fonctions du *Serveur*. Elle contient la liste des *Profiles* OPC UA pris en charge par le *Serveur*. Les *Profiles* OPC UA sont définis dans la CEI 62541-7.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les *Certificats* supplémentaires émis par d'autres organisations peuvent être inclus pour identifier des fonctions supplémentaires du *Serveur*. Les exemples de ces *Profiles* incluent la prise en charge pour des modèles spécifiques d'informations et la prise en charge pour l'accès aux types spécifiques de dispositifs.

Lorsqu'une *Session* est créée, le *Serveur* ajoute une entrée pour le *Client* dans sa *Variable SessionDiagnosticArray*. Voir la CEI 62541-5 pour la description de cette *Variable*.

Les *Sessions* sont créées de façon à être indépendantes de la connexion de communications sous-jacente. Par conséquent, une défaillance de la connexion de communications n'affecte pas immédiatement la *Session*. Le mécanisme de reprise sur une erreur de la connexion de communications dépend de la mise en correspondance de SecureChannel décrite dans la CEI 62541-6.

Les *Sessions* sont automatiquement fermées par le *Serveur* si le *Client* oublie d'émettre une demande de *Service* dans la période d'attente de temporisation négociée avec le *Serveur* dans la réponse à la demande de *Service CreateSession*. Le *Serveur* est ainsi protégé des défaillances du *Client* et des situations dans lesquelles la connexion sous-jacente défectueuse ne peut pas être rétablie. Les *Clients* doivent être prêts à transmettre des demandes dans le temps prévu afin d'empêcher la fermeture automatique de cette *Session*. Les *Clients* peuvent explicitement mettre fin à des *Sessions* au moyen du *ServiceCloseSession* (Service Fermer la Session).

Lorsqu'il est mis fin à une *Session*, toutes les demandes en cours sur cette *Session* sont abandonnées et des *StatusCodes Bad_SessionClosed* sont retournés au *Client*. De plus, le *Serveur* supprime de sa *Variable SessionDiagnosticArray* (Matrice de diagnostic de session) l'entrée pour le *Client* et en notifie les autres éventuels *Clients* qui s'étaient abonnés à cette entrée.

Si un *Client* invoque le *Service CloseSession*, tous les *Souscriptions* (Abonnements) associés à la *Session* sont également supprimées si le fanion *deleteSubscriptions* (Effacer les abonnements) est mis à TRUE. Si un *Server* met fin à une *Session* pour une autre raison quelconque, les *Souscriptions* associés à cette *Session* ne sont pas supprimés. Chaque *Souscription* a sa propre durée de vie pour se protéger des pertes de données en cas de fermeture d'une *Session*. Dans ces cas, la *Souscription* peut être réaffectée à un autre *Client* avant que sa durée de vie n'expire.

Certains *Serveurs*, tels que les *Serveurs* combinés, agissent aussi comme *Clients* pour d'autres *Serveurs*. Les *Serveurs* prennent typiquement en charge plus d'un utilisateur système, agissant comme leurs agents auprès des *Serveurs* qu'ils représentent. La sécurité pour ces *Serveurs* est prise en charge à deux niveaux.

Premièrement, chaque demande de *Service* OPC UA contient un paramètre chaîne qui est utilisé pour contenir un id d'enregistrement d'audit. Un *Client* ou tout *Serveur* opérant comme *Client*, tel qu'un *Serveur* combiné par exemple, peut créer une entrée locale de journal d'audits pour une demande qu'il présente. Ce paramètre permet au *Client* de transmettre l'identificateur pour cette entrée avec la demande. Si le *Serveur* aussi maintient un journal d'audits, il peut inclure cet id dans l'entrée de journal d'audits qu'il écrit. Lorsque le journal est examiné et l'entrée trouvée, l'examinateur pourra la relier directement à l'entrée de journal d'audits créée par le *Client*. Cette fonction permet la traçabilité à travers les journaux d'audits dans un système. Voir la CEI/TR 62541-2 pour des informations complémentaires relatives aux audits. Un *Serveur* qui maintient un journal d'audits doit fournir les informations dans les entrées du journal d'audits via des *Messages* d'événements définis dans la présente norme. Le *Serveur* peut choisir de fournir uniquement les informations d'*Audit* via des *Messages* d'événements. L'*EventType Audit* est défini dans la CEI 62541-3.

Deuxièmement, ces *Serveurs* combinés peuvent ouvrir des *Sessions* indépendantes sur les *Serveurs* sous-jacents pour chaque *Client* qui, à partir d'eux, accède aux données. La Figure 14 illustre ce concept.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

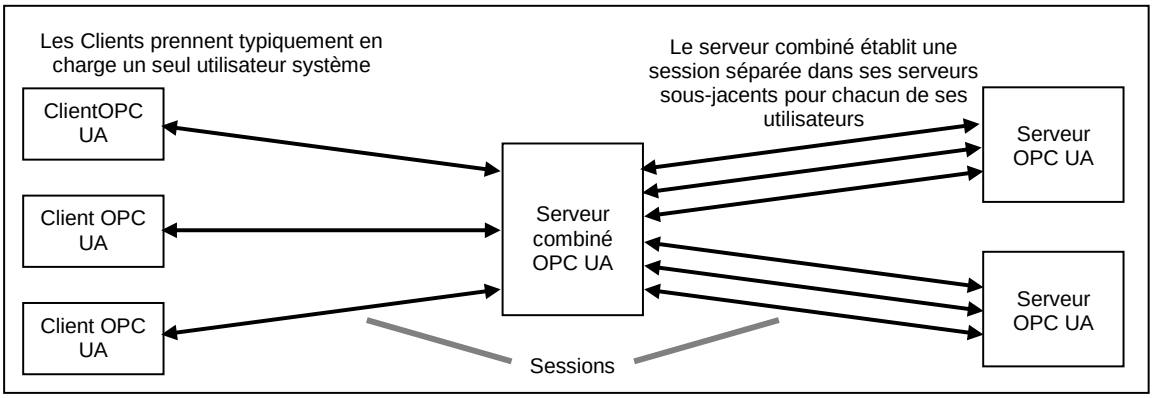


Figure 14 – Multiplexage d'utilisateurs sur une Session

5.6.2.2 Paramètres

Le Tableau 11 définit les paramètres pour le *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 11 – Paramètres du service CreateSession

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. L'*authenticationToken* (jeton d'authentification) est toujours omis. Le type *RequestHeader* est défini en 7.26.
clientDescription	Application Description	Informations qui décrivent l'application *Client*. Le type *ApplicationDescription* est défini en 7.1.
serverUri	String	Cette valeur est seulement spécifiée si l'*EndpointDescription* a un *gatewayServerUri*. Cette valeur est l'*applicationUri* issue de l'*EndpointDescription* qui est l'*applicationUri* pour le *Serveur* sous-jacent. Le type *EndpointDescription* est défini en 7.9.
endpointUrl	String	L'adresse réseau que le *Client* a utilisée pour accéder au *Session Endpoint*. Il convient que la partie *HostName* de l'URL soit l'un des *HostNames* pour l'application qui sont spécifiés dans l'*ApplicationInstanceCertificate* du *Serveur* (voir 7.2). Le *Serveur* doit soulever un événement *AuditUrlMismatchEventType* (un événement de type événement de discordance d'URL d'audit) si l'URL ne correspond pas aux *HostNames* du serveur. Le type d'événement *AuditUrlMismatchEventType* est défini dans la CEI 62541-5. Le serveur utilise cette information pour des diagnostics et pour déterminer le jeu d'*EndpointDescriptions* à retourner dans la réponse.
sessionName	String	Chaîne lisible par l'homme qui identifie une *Session*. Le *Serveur* rend ce nom et le *sessionId* visibles dans son *AddressSpace* à des fins de diagnostic. Il convient que le *Client* fournisse un nom qui est unique pour l'instance du *Client*. Si ce paramètre n'est pas spécifié, le *Serveur* doit attribuer une valeur.
clientNonce	ByteString	Un nombre aléatoire qu'il convient de ne jamais utiliser dans aucune autre demande. Ce nombre doit avoir une longueur minimale de 32 octets. Les profils peuvent augmenter la longueur requise. Le *Serveur* doit utiliser cette valeur pour prouver la possession de son Certificat d'instance d'application dans la réponse.
clientCertificate	ApplicationInstance Certificate	Le Certificat d'instance d'application émis au *Client*. Le type *ApplicationInstanceCertificate* est défini en 7.2.
Réclamée SessionTimeout	Duration	Nombre maximum demandé de millisecondes pendant lesquelles il convient qu'une *Session* reste ouverte sans activité. Si le *Client* oulie d'émettre une demande de Service dans la limite de cet intervalle, le *Serveur* doit automatiquement mettre fin à la *Session* du *Client*.
maxResponse MessageSize	UInt32	La taille maximale, en octets, pour le corps de tout message de réponse. Il convient que le *Serveur* retourne un défaut de service *Bad_ResponseTooLarge* si un message de réponse dépasse cette limite. La valeur zéro indique que ce paramètre n'est pas utilisé. Le paragraphe 5.3 fournit plus d'informations sur l'utilisation de ce paramètre.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour le type *ResponseHeader*).
sessionId	NodeId	Un *NodeId* unique affecté par le *Serveur* à une *Session*. Cet identificateur est utilisé pour accéder aux informations de diagnostic pour une *Session* dans l'Espace d'Adresses du *Serveur*. Il est également utilisé dans les journaux d'audits et les éventuels événements qui rapportent des informations relatives à cette *Session*. Les informations de diagnostic d'une *Session* sont décrites dans la CEI 62541-5. Les journaux d'audits et leurs événements connexes sont décrits en 6.2
authentification Token	Session AuthenticationToken	Un identificateur unique affecté par le *Serveur* à la *Session*. Cet identificateur doit être transmis dans le *RequestHeader* de chaque demande et il est utilisé avec le *SecureChannelId* pour déterminer si, oui ou non, un *Client* a accès à la *Session*. Cet identificateur ne doit pas être réutilisé d'une manière induisant le risque pour le *Client* ou le *Serveur* de le confondre avec une *Session* précédente ou existante. Le type *SessionAuthenticationToken* est décrit en 7.29.
revisedSession Timeout	Duration	Nombre maximum effectif de millisecondes pendant lesquelles une *Session* doit rester ouverte sans activité. Il convient que le *Serveur* tente d'honorer la demande du *Client* pour ce paramètre, mais il peut négocier cette valeur à la hausse ou à la baisse pour satisfaire à ses propres contraintes.
serverNonce	ByteString	Un nombre aléatoire qu'il convient de ne jamais utiliser dans aucune autre demande. Ce nombre doit avoir une longueur minimale de 32 octets. Le *Client* doit utiliser cette valeur pour prouver la possession de son Certificat d'instance d'application dans la demande *ActivateSession*. Cette valeur peut aussi être utilisée pour prouver la possession du *userIdentityToken* qu'il a spécifié dans la demande *ActivateSession*.
serverCertificate	ApplicationInstance Certificate	Le Certificat d'instance d'application émis au *Serveur*. Un *Serveur* doit prouver la possession en utilisant la clé privée pour signer le *Nonce* fourni par le *Client* dans la demande. Le *Client* doit vérifier que ce Certificat est le même que celui qu'il a utilisé pour créer le *Canal Sécurisé* . Le type *ApplicationInstanceCertificate* est défini en 7.2.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
serverEndpoints []	Endpoint Description	Liste des *Endpoints* que le serveur prend en charge. Le *Serveur* doit retourner un jeu d'*EndpointDescriptions* disponibles pour le *serverUri* spécifié dans la demande. Le type *EndpointDescription* est défini en 7.9. Le *Client* doit vérifier cette liste avec la liste issue d'un *Discovery Endpoint* s'il a utilisé un *Discovery Endpoint* pour rechercher des *EndpointDescriptions*.
serverSoftware Certificates []	SignedSoftware Certificate	Ce sont les *SoftwareCertificates* qui ont été émis à l'application *Serveur*. Le *productUri* contenu dans les *SoftwareCertificates* doit correspondre au *productUri* dans l'*EndpointDescription* utilisée par le *Client* pour se connecter au *Serveur*. Il convient d'ignorer les Certificats sans *productUris* correspondants. Il convient que les Clients appellent le service *CloseSession* s'ils ne sont pas satisfaits des *SoftwareCertificates* fournis par le *Serveur*. Le type *SignedSoftwareCertificate* est défini en 7.31.
serverSignature	SignatureData	Il s'agit d'une signature générée par la clé privée associée au *serverCertificate*. Ce paramètre est calculé en accolant le *clientNonce* au *clientCertificate* et en signant la séquence d'octets obtenue. Le *SignatureAlgorithm* (algorithme de signature) doit être l'algorithme d'*asymmetricSignature* (de signature asymétrique) spécifié dans la *SecurityPolicy* pour l'*Endpoint*. Le type *SignatureData* est défini en 7.30.
maxRequest MessageSize	UInt32	La taille maximale, en octets, pour le corps de tout message de demande. Il convient que la *Communication Stack* du *Client* retourne une erreur *Bad_RequestTooLarge* à l'application si un message de demande dépasse cette limite. La valeur zéro indique que ce paramètre n'est pas utilisé. Le paragraphe 5.3 fournit plus d'informations sur l'utilisation de ce paramètre.

5.6.2.3 Résultats des services

Le Tableau 12 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 12 – Codes des résultats de services CreateSession

Id symbolique	Description
Bad_SecureChannelIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NonceInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SecurityChecksFailed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateTimeInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerTimeInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateHostNameInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUriInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUseNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerUseNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateUntrusted	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateRevocationUnknown	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerRevocationUnknown	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateRevoked	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_CertificateIssuerRevoked	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManySessions	Le serveur a atteint le nombre maximal de sessions.
Bad_ServerUriInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.6.3 Activer une session (ActivateSession)

5.6.3.1 Description

Ce *Service* est utilisé par le *Client* pour présenter ses *SoftwareCertificates* au *Serveur* pour validation et pour spécifier l'identité de l'utilisateur associé à la *Session*. Cette demande de *Service* doit être émise par le *Client* avant qu'il n'émette une quelconque autre demande de *Service* après *CreateSession*. Si tel n'est pas le cas, le *Serveur* doit fermer la *Session*.

Chaque fois que le *Client* appelle ce *Service*, le *Client* doit prouver qu'il s'agit de la même application qui a appelé le *Service CreateSession*. Pour ce faire, le *Client* crée une signature avec la clé privée associée au *clientCertificate* spécifié dans la demande *CreateSession*. Cette signature est créée en accolant le dernier *serverNonce* fourni par le *Serveur* au *serverCertificate* et en calculant la signature de la séquence d'octets obtenue.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Une fois utilisé, un *serverNonce* ne peut plus être réutilisé. C'est pourquoi le *Serveur* retourne un nouveau *serverNonce* chaque fois que le *Service ActivateSession* est appelé.

Lorsque le *Service ActivateSession* est appelé la première fois, le serveur doit rejeter la demande si le *Canal Sécurisé* n'est pas la même que celle qui est associée à la demande *CreateSession*. Les appels consécutifs à *ActivateSession* peuvent être associés à des *Canal Sécurisés* différents. Si tel est le cas, le *Serveur* doit vérifier que le *Certificat* utilisé par le *Client* pour créer le nouveau *Canal Sécurisé* est le même que le *Certificat* utilisé pour créer le *Canal Sécurisé* d'origine. En outre, le *Serveur* doit vérifier que le *Client* a fourni un *UserIdentityToken* identique au jeton qui a ouvert la *Session*. Une fois que le *Serveur* a accepté le nouveau *Canal Sécurisé*, il doit rejeter les demandes envoyées par l'ancien *Canal Sécurisé*.

Le *Service ActivateSession* est utilisé pour associer une identité d'utilisateur à une *Session*. Lorsqu'un *Client* fournit une identité d'utilisateur, il doit fournir la preuve qu'il est habilité à utiliser l'identité d'utilisateur en question. Le mécanisme exact utilisé pour fournir cette preuve dépend du type de *UserIdentityToken*. Si le jeton est un *UserNameIdentityToken*, la preuve est le *password* (Mot de passe) qui a été inclus dans le jeton. Si le jeton est un X509IdentityToken, la preuve est une signature générée avec la clé privée associée au *Certificate*. Les données à signer sont créées en accolant le dernier *serverNonce* au *serverCertificate* spécifié dans la réponse *CreateSession*. Si un jeton inclut un secret, il convient de le déchiffrer à l'aide de la clé publique contenue dans le *serverCertificate.*

Les *Clients* peuvent modifier l'identité d'un utilisateur associé à une *Session* en appelant le *Service ActivateSession*. Le *Serveur* valide les signatures fournies avec la demande puis valide la nouvelle identité d'utilisateur. En l'absence d'erreurs, le *Serveur* remplace l'identité d'utilisateur pour la *Session*. La modification de l'identité utilisateur pour une *Session* peut induire des discontinuités des *Souscriptions* actifs car le *Serveur* peut devoir casser des connexions au système sous-jacent et les rétablir en utilisant les nouveaux authentifiants.

Lorsqu'un *Client* fournit une liste d'identificateurs de paramètres de lieu dans la demande, il est requis de chaque identificateur de paramètre de lieu qu'il contienne la composante langue. Il peut facultativement contenir la composante <country/region> (pays/région). Lorsque le *Serveur* retourne la réponse, il peut aussi retourner les deux composantes langue et pays/région ou juste la composante langue comme la valeur par défaut de son identificateur de paramètre de lieu.

Lorsqu'un *Serveur* retourne une chaîne au *Client*, il détermine d'abord s'il existe des traductions pour celle-ci. S'il en existe, le *Serveur* retourne la chaîne dont l'identificateur de paramètre de lieu correspond exactement à l'identificateur de paramètre de lieu ayant la priorité la plus haute dans la liste fournie par le *Client*.

S'il n'existe pas de correspondances exactes, le *Serveur* ignore la composante <country/region> de l'identificateur de paramètre de lieu et retourne la chaîne dont la composante <language> correspond à la composante <language> de l'identificateur de paramètre de lieu ayant la priorité la plus haute dans la liste fournie par le *Client*.

S'il n'y a pas de correspondances, le *Serveur* retourne la chaîne qu'il a avec l'identificateur de paramètre de lieu.

Un *Gateway Server* est censé se substituer à l'utilisateur donné par le *Client* lorsqu'il se connecte au *Serveur* sous-jacent. Cela signifie qu'il doit recalculer les signatures sur l'*UserIdentityToken* en utilisant les nonces fournis par le *Serveur* sous-jacent. Le *Gateway Serveur* aura à utiliser ses propres authentifiants d'utilisateur si le *UserIdentityToken* fournit par le *Client* ne prend pas en charge la substitution d'identité.

5.6.3.2 Paramètres

Le Tableau 13 définit les paramètres pour le *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 13 – Paramètres du service ActivateSession

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes. Le type *RequestHeader* est défini en 7.26.
clientSignature	SignatureData	Il s'agit d'une signature générée par la clé privée associée au *clientCertificate*. Le *SignatureAlgorithm* (algorithme de signature) doit être l'algorithme d'*asymmetricSignature* (de signature asymétrique) spécifié dans la *SecurityPolicy* pour l'*Endpoint*. Le type *SignatureData* est défini en 7.30.
clientSoftwareCertificates []	SignedSoftware Certificate	Ce sont les *SoftwareCertificates* qui ont été émis à l'application *Client*. Le *productUri* contenu dans les *SoftwareCertificates* doit correspondre au *productUri* dans l'Application*Description* utilisée par le *Client* dans les demandes *CreateSession*. Il convient d'ignorer les *Certificates* sans *productUri*s correspondants. Les *Serveurs* peuvent rejeter des connexions issues des *Clients* s'ils ne sont pas satisfaits par les *SoftwareCertificates* fournis par le *Client*. Il n'est nécessaire de spécifier ce paramètre que pendant le premier appel au *Service ActivateSession* au cours d'une seule *Session* d'application. Le type *SignedSoftwareCertificate* est défini en 7.31.
localeIds []	LocaleId	Liste des identificateurs de paramètres de lieu dans l'ordre des priorités pour les chaînes localisées. Le premier *localeId* dans la liste a la priorité la plus haute. Si le *Serveur* retourne une chaîne localisée au *Client*, le *Serveur* doit retourner la traduction avec la plus haute priorité qu'il peut. S'il n'a de traduction pour aucun des paramètres de lieu identifiés dans cette liste, il doit retourner la valeur chaîne qu'il a et inclure l'identificateur de paramètre de lieu avec la chaîne. Voir la CEI 62541-3 pour plus de détails sur les identificateurs de paramètres de lieu. Si le *Client* ne réussit pas à spécifier au moins un identificateur de paramètre de lieu, le *Serveur* doit utiliser n'importe lequel en sa possession. Il n'est nécessaire de spécifier ce paramètre que pendant le premier appel au *Service ActivateSession* au cours d'une seule *Session* d'application. S'il n'est pas spécifié, le *Serveur* doit continuer d'utiliser les *localeIds* courants pour la *Session*.
userIdentityToken	Extensible Parameter UserIdentityToken	Les authentifiants de l'utilisateur associé à l'application *Client*. Le *Serveur* utilise ces authentifiants pour déterminer s'il convient ou non d'autoriser le *Client* à activer une *Session* et pour déterminer également les ressources auxquelles le *Client* a accès au cours de la *Session*. Le *UserIdentityToken* est un type de paramètre extensible défini en 7.35. L'EndpointDescription spécifie les *UserIdentityTokens* que le *Serveur* doit accepter.
userTokenSignature	SignatureData	Si le *Client* a spécifié un jeton d'identité d'utilisateur qui prend en charge les signatures numériques, il doit créer une signature et la passer comme étant ce paramètre. Autrement le paramètre est omis. Le *SignatureAlgorithm* dépend du type de jeton d'identité. Le type *SignatureData* est défini en 7.30.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
serverNonce	ByteString	Un nombre aléatoire qu'il convient de ne jamais utiliser dans aucune autre demande. Ce nombre doit avoir une longueur minimale de 32 octets. Le *Client* doit utiliser cette valeur pour prouver la possession de son *Certificate* d'instance d'application lors de la prochaine demande d'appel à *ActivateSession*.
results []	StatusCode	Liste de résultats de validation pour les *SoftwareCertificates* (voir 7.33 pour la définition de *StatusCode*).
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic associées à des erreurs de validation de *SoftwareCertificate* (voir 7.8 pour la définition de *DiagnosticInfo*). Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.6.3.3 Résultats des services

Le Tableau 14 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 14 – Codes des résultats de services ActivateSession

Id symbolique	Description
Bad_IdentityTokenInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_IdentityTokenRejected	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ApplicationSignatureInvalid	La signature fournie par l'application cliente est absente ou non valide.
Bad_UserSignatureInvalid	La signature du jeton d'utilisateur est absente ou non valide.
Bad_NoValidCertificates	Le *Client* n'a pas fourni au moins un certificat de logiciel qui est valide et satisfait aux exigences relatives aux profils pour le *Serveur*.
Bad_IdentityChangeNotSupported	Le *Serveur* ne prend pas en charge la modification de l'identité d'utilisateur affectée à la session.

5.6.4 Fermer une session (CloseSession)

5.6.4.1 Description

Ce *Service* est utilisé pour mettre fin à une *Session*. Le *Serveur* prend les actions suivantes lorsqu'il reçoit une demande *CloseSession*:

a) Il arrête d'accepter des demandes pour cette *Session*. Toutes les demandes ultérieures reçues pour la *Session* sont rejetées.

b) Il retourne des réponses négatives avec le *StatusCode* Bad_SessionClosed à toutes les demandes qui sont actuellement en cours pour permettre le retour en temps utile de la réponse *CloseSession*. Les *Clients* sont vivement encouragés à attendre que toutes les demandes en cours se terminent avant de présenter une demande *CloseSession*.

c) Il retire l'entrée pour le *Client* dans sa *Variable SessionDiagnosticArray*.

5.6.4.2 Paramètres

Le Tableau 15 définit les paramètres pour le *Service*.

Tableau 15 – Paramètres du service CloseSession

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
deleteSubscriptions	Boolean	Si la valeur est TRUE, le *Serveur* supprime tous les Abonnements associés à la Session. Si la valeur est FALSE, le *Serveur* conserve les Abonnements associés à la Session jusqu'à ce qu'ils expirent sur la base de leur propre durée de vie.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).

5.6.4.3 Résultats des services

Le Tableau 16 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 16 – Codes des résultats de services CloseSession

Id symbolique	Description
Bad_SessionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.6.5 Annuler (Cancel)

5.6.5.1 Description

Ce *Service* est utilisé pour annuler des demandes de Services en cours. Les demandes de services qui ont été annulées avec succès doivent répondre avec Bad_RequestCancelledByClient.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.6.5.2 Paramètres

Le Tableau 17 définit les paramètres pour le *Service*.

Tableau 17 – Paramètres du service Cancel

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
requestHandle	IntegerId	Le *requestHandle* (descripteur de demande) affecté à une ou plusieurs demandes qu'il convient d'annuler. Toutes les demandes en cours avec les *requestHandles* correspondants doivent être annulées.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
cancelCount	UInt32	Nombre de demandes annulées.

5.6.5.3 Résultats des services

Les *StatusCodes* communs sont définis au Tableau 165.

5.7 Jeu de services de gestion de nœuds (NodeManagement Service Set)

5.7.1 Vue générale

Ce *Service Set* (Jeu de services) définit les *Services* pour ajouter et supprimer des Noeuds de *l'Espace d'Adresses* et les *Références* entre eux. Tous les *Noeuds* continuent d'exister dans l'Espace d'Adresses même si le *Client* qui les a créés se déconnecte du *Serveur*.

5.7.2 Ajouter des nœuds (AddNodes)

5.7.2.1 Description

Ce *Service* est utilisé pour ajouter un ou plusieurs *Noeuds* dans la hiérarchie de l'Espace d'Adresses. En utilisant ce *Service*, chaque Noeud est ajouté comme étant le *TargetNode* d'une *HierarchicalReference* pour assurer que *l'Espace d'Adresses* est pleinement connecté et que le Noeud est ajouté comme enfant dans la hiérarchie de *l'Espace d'Adresses* (voir la CEI 62541-3).

5.7.2.2 Paramètres

Le Tableau 18 définit les paramètres pour le *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 18 – Paramètres du service AddNodes

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
nodesToAdd []	AddNodesItem	Liste des Noeuds à ajouter. Tous les Noeuds sont ajoutés comme une *Référence* à un Noeud existant en utilisant un *ReferenceType* hiérarchique.
parentNodeId	Expanded NodeId	*ExpandedNodeId* (identificateur de nœud étendu) du Noeud parent pour la *Référence*. Le type *ExpandedNodeId* est défini en 7.10.
referenceTypeId	NodeId	*NodeId* du *ReferenceType* hiérarchique à utiliser pour la *Référence* allant du Noeud parent au nouveau Noeud.
requestedNewNodeId	Expanded NodeId	*NodeId* étendu demandé par le *Client* pour le Noeud à ajouter. Le *serverIndex* dans le NodeId étendu doit être 0. Si le *Serveur* ne peut pas utiliser ce *NodeId*, il rejette ce Noeud et renvoie un code d'erreur approprié. Si le *Client* ne souhaite pas demander un *NodeId*, il met la valeur de ce paramètre au *NodeId* étendu null. Si le Noeud à ajouter est un Noeud de *ReferenceType*, il convient que son *NodeId* soit un identificateur numérique. Voir la CEI 62541-3 pour une description des *NodeIds* de *ReferenceType*.
browseName	QualifiedName	Le nom de navigation du Noeud à ajouter.
nodeClass	NodeClass	*NodeClass* (classe de nœuds) du Noeud à ajouter.
nodeAttributes	Extensible Parameter NodeAttributes	Les *Attributs* qui sont spécifiques à la *NodeClass*. Le type de paramètre *NodeAttributes* est un type de paramètre extensible spécifié en 7.18. Un *Client* est autorisé à omettre les valeurs pour certains ou tous les *Attributs*. Si une valeur d'*Attribut* est omise, le *Serveur* doit utiliser les valeurs par défaut provenant du *TypeDefinitionNode*. Si un *TypeDefinitionNode* n'a pas été fourni, le *Serveur* doit choisir une valeur par défaut appropriée. Le *Serveur* peut encore ajouter un Attribut facultatif au Noeud avec une valeur par défaut appropriée même si le *Client* ne spécifie pas une valeur.
typeDefinition	Expanded NodeId	*NodeId* du *TypeDefinitionNode* pour le Noeud à ajouter. Ce paramètre doit être null pour toutes les *NodeClasses* autres que *Object* et *Variable* et, dans ce cas, il doit être fourni.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	AddNodesResult	Liste de résultats pour les Noeuds à ajouter. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToAdd*
statusCode	StatusCode	*StatusCode* pour le Noeud à ajouter (voir 7.33 pour la définition de *StatusCode*).
addedNodeId	NodeId	*NodeId* affecté par le *Serveur* au Noeud ajouté. *NodeId* Null si l'opération a échoué.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les Noeuds à ajouter (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToAdd*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.7.2.3 Résultats des services

Le Tableau 19 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 19 – Codes des résultats de services AddNodes

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.7.2.4 StatusCodes

Le Tableau 20 définit les valeurs du paramètre *statusCode* de niveau opération qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 20 – Codes de résultats de niveau opération des AddNodes

Id symbolique	Description
Bad_ParentNodeIdInvalid	L'identificateur de nœud parent ne renvoie pas à un nœud valide.
Bad_ReferenceTypeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceNotAllowed	La référence n'a pas pu être créée car elle viole des contraintes imposées par le modèle de données.
Bad_NodeIdRejected	L'identificateur de nœud demandé a été rejeté car il était invalide ou parce que le serveur n'autorise pas que des identificateurs de nœuds soient spécifiés par le client.
Bad_NodeIdExists	L'identificateur de nœud demandé est déjà utilisé par un autre nœud.
Bad_NodeClassInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_BrowseNameInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_BrowseNameDuplicated	Le nom de navigation n'est pas unique parmi les nœuds qui partagent la même relation avec le parent.
Bad_NodeAttributesInvalid	Les *Attributs* de nœud ne sont pas valides pour la classe de nœuds.
Bad_TypeDefinitionInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.

5.7.3 Ajouter des références (AddReferences)

5.7.3.1 Description

Ce *Service* est utilisé pour ajouter une ou plusieurs *Références* à un ou plusieurs Noeuds. La *NodeClass* est un paramètre d'entrée qui est utilisé pour valider le fait que la *Référence* à ajouter corresponde à la *NodeClass* du *TargetNode*. Ce paramètre n'est pas validé si la *Référence* renvoie à un *TargetNode* dans un *Serveur* distant.

Dans certains cas, l'ajout de nouvelles *Références* à l'Espace d'Adresses exige que le *Serveur* ajoute de nouveaux identificateurs de *Serveur* à la *Variable ServerTable* (Tableau des serveurs) du *Serveur*. C'est pourquoi les *Serveurs* distants sont identifiés par leur URI et non par leur indice de *ServerTable*. Le *Serveur* peut ainsi ajouter les URI de *Serveurs* distants à son *ServerTable*.

5.7.3.2 Paramètres

Le Tableau 21 définit les paramètres pour le *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 21 – Paramètres du service AddReferences

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
referencesToAdd []	AddReferences Item	Liste d'instances *Référence* à ajouter au *SourceNode*. La *targetNodeClass* de chaque *Référence* dans la liste doit correspondre à la *NodeClass* du *TargetNode*.
sourceNodeId	NodeId	*NodeId* au Noeud auquel la *Référence* est à ajouter. Le Noeud source doit toujours exister dans le *Serveur* pour ajouter la *Référence*. Le paramètre isForward («est direct») peut être mis à FALSE si le Noeud cible est sur le *Serveur* local et le Noeud source sur le *Serveur* distant.
referenceTypeId	NodeId	*NodeId* du *ReferenceType* qui définit la *Référence*.
isForward	Boolean	Si la valeur est TRUE, le serveur crée une Référence directe (forward Reference). Si la valeur est FALSE, le serveur crée une Référence inverse (inverse Reference).
targetServerUri	String	URI du *Serveur* distant. Si ce paramètre n'est pas null, il se substitue au *serverIndex* dans le *targetNodeId*.
targetNodeId	Expanded NodeId	*NodeId* étendu du *TargetNode*. Le type *ExpandedNodeId* est défini en 7.10.
targetNodeClass	NodeClass	*NodeClass* du *TargetNode*. Le *Client* doit spécifier cela car le *TargetNode* pourrait ne pas être directement accessible au *Serveur*.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *Références* à ajouter (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *referencesToAdd.*
diagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les *Références* à ajouter (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *referencesToAdd.* Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.7.3.3 Résultats des services

Le Tableau 22 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 22 – Codes des résultats de services AddReferences

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.7.3.4 StatusCodes

Le Tableau 23 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 23 – Codes de résultats de niveau opération des AddReferences

Id symbolique	Description
Bad_SourceNodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceTypeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ServerUriInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TargetNodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeClassInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceNotAllowed	La référence n'a pas pu être créée car elle viole des contraintes imposées par le modèle de données à ce serveur.
Bad_ReferenceLocalOnly	Le type de référence n'est pas valide pour une référence à un *Serveur* distant.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_DuplicateReferenceNotAllowed	Le type de référence entre les nœuds est déjà défini.
Bad_InvalidSelfReference	Le serveur n'autorise pas ce type d'autoréférence sur ce nœud.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.7.4 Supprimer des noeuds (DeleteNodes)

5.7.4.1 Description

Ce *Service* est utilisé pour supprimer un ou plusieurs *Noeuds* de *l'Espace d'Adresses*.

Lorsque l'un quelconque des *Noeuds* supprimés par une invocation de ce *Service* est le *TargetNode* d'une *Référence*, ces *Références* sont laissées non résolues sur la base du paramètre *deleteTargetReferences*.

Lorsque l'un quelconque des *Noeuds* supprimés par une invocation de ce *Service* est sous surveillance, une *Notification* contenant le statut code Bad_NodeIdUnknown est envoyée au *Client* sous surveillance indiquant que le *Noeud* a été supprimé.

5.7.4.2 Paramètres

Le Tableau 24 définit les paramètres pour le *Service*.

Tableau 24 – Paramètres du service DeleteNodes

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
nodesToDelete []	DeleteNodes Item	Liste des Noeuds à supprimer.
nodeId	NodeId	*NodeId* du Noeud à supprimer.
deleteTargetReference	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE supprimer les *Références* dans les *TargetNodes* qui *référencent* le Noeud à supprimer. FALSE supprimer seulement les *Références* dont le Noeud à supprimer est la source. Le *Serveur* ne peut pas garantir qu'il est capable de supprimer toutes les *Références* cibles si ce paramètre est TRUE.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *Noeuds* à supprimer (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToDelete.*
diagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les *Noeuds* à supprimer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToDelete.* Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.7.4.3 Résultats des services

Le Tableau 25 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 25 – Codes de résultats des services DeleteNodes

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.7.4.4 StatusCodes

Le Tableau 26 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 26 – Codes de résultats de niveau opération des DeleteNodes

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NoDeleteRights	Voir le Tableau 166 pour la description de ce code de résultat.
Uncertain_ReferenceNotDeleted	Le serveur n'était pas capable de supprimer toutes les références cibles.

5.7.5 Supprimer des références (DeleteReferences)

5.7.5.1 Description

Ce *Service* est utilisé pour supprimer une ou plusieurs *Références* d'un Noeud.

Lorsqu'un nombre quelconque de *Références* supprimées par une invocation de ce *Service* est contenu dans une Vue, la *Property ViewVersion* est mise à jour si cette *Property* est prise en charge.

La suppression d'une Référence doit déclencher un *Evénement ModelChange* (Changement de modèle).

5.7.5.2 Paramètres

Le Tableau 24 définit les paramètres du *Service*.

Tableau 27 – Paramètres du service DeleteReferences

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
referencesToDelete []	DeleteReferences Item	Liste des *Références* à supprimer.
sourceNodeId	NodeId	*NodeId* du Noeud qui contient la *Référence* à supprimer.
referenceTypeId	NodeId	*NodeId* du *ReferenceType* qui définit la *Référence* à supprimer.
isForward	Boolean	Si la valeur est TRUE, le serveur supprime une Référence directe (forward Reference). Si la valeur est FALSE, le serveur supprime une Référence inverse (inverse Reference).
targetNodeId	ExpandedNodeId	*NodeId* du *TargetNode* de la *Référence*. Si l'indice du *Serveur* indique que le *TargetNode* est un Noeud distant, le *nodeId* doit contenir l'URI d'espace de noms absolu. Si le *TargetNode* est un Noeud local, le *nodeId* doit contenir l'indice d'espace de noms.
deleteBidirectional	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE supprimer la *Référence* spécifiée et la *Référence* opposée partant du *TargetNode*. Si le *TargetNode* est situé sur un *Serveur* distant, le *Serveur* est autorisé à supprimer la *Référence* spécifiée seulement. FALSE supprimer la *Référence* spécifiée seulement.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *Références* à supprimer (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *referencesToDelete.*
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *Références* à supprimer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *referencesToDelete.* Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été trouvée dans le traitement de la demande.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.7.5.3 Résultats des services

Le Tableau 28 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 28 – Codes de résultats des services DeleteReferences

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.7.5.4 StatusCodes

Le Tableau 29 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 29 – Codes de résultats de niveau opération des DeleteReferences

Id symbolique	Description
Bad_SourceNodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceTypeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ServerIndexInvalid	L'indice de serveur n'est pas valide.
Bad_TargetNodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NoDeleteRights	Voir le Tableau 166 pour la description de ce code de résultat.

5.8 Jeu de services pour les vues (View Service Set)

5.8.1 Vue d'ensemble

Les *Clients* utilisent les *Services* navigation du *View Service Set* (Jeu de services pour les vues) pour naviguer à travers l'Espace d'Adresses ou à travers une Vue qui est un sous-ensemble de *l'Espace d'Adresses*.

Une *Vue* est un sous-ensemble de *l'Espace d'Adresses* créé par le *Serveur*. Les futures versions de la présente norme peuvent aussi définir des services pour créer des *Vues* définies par le *Client*. Voir la CEI 62541-5 pour une description de l'organisation des vues dans *l'Espace d'Adresses*.

5.8.2 Parcourir (Browse)

5.8.2.1 Description

Ce *Service* est utilisé pour découvrir les *Références* d'un *Noeud* spécifié. La navigation peut en outre être limitée par l'utilisation d'une Vue. Ce *Service* Browse prend aussi en charge une fonction de filtrage primitive

5.8.2.2 Paramètres

Le Tableau 30 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 30 – Paramètres du service Browse

<table>
<tr><th>Nom</th><th>Type</th><th>Description</th></tr>
<tr><td>Request</td><td></td><td></td></tr>
<tr><td>requestHeader</td><td>RequestHeader</td><td>Paramètres communs aux demandes (voir 7.26 pour la définition de RequestHeader).</td></tr>
<tr><td>view</td><td>ViewDescription</td><td>Description de la Vue à parcourir (voir 7.37 pour la définition de ViewDescription). Une valeur vide de ViewDescription indique l'Espace d'Adresses tout entier. L'utilisation de la valeur vide de ViewDescription entraîne que toutes les Références du nodeToBrowse sont retournées. L'utilisation d'une autre Vue entraîne que seules les Références du nodeToBrowse qui sont définies pour la Vue en question seront retournées.</td></tr>
<tr><td>requestedMax ReferencesPerNode</td><td>Counter</td><td>Indique le nombre maximal de références à retourner pour chaque Noeud de départ spécifié dans la demande. La valeur 0 indique que le Client n'impose aucune limitation (voir 7.5 pour la définition de Counter).</td></tr>
<tr><td>nodesToBrowse []</td><td>BrowseDescription</td><td>Une liste de nœuds à parcourir (Browse)</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>NodeId du Noeud à parcourir. Le nodeToBrowse passé doit faire partie de la Vue passée.</td></tr>
<tr><td>browseDirection</td><td>enum BrowseDirection</td><td>Une énumération qui spécifie le sens des Références à suivre. Cette énumération a les valeurs suivantes:
FORWARD_0 sélectionner des Références directes seulement.
INVERSE_1 sélectionner des Références inverses seulement.
BOTH_2 sélectionner des Références directes et inverses.
Les Références retournées indiquent effectivement le sens que le Serveur a suivi dans le paramètre isForward de la ReferenceDescription.
Les Références symétriques sont toujours considérées être dans le sens direct; par conséquent, le fanion isForward est toujours mis à TRUE et les Références symétriques ne sont pas retournées si browseDirection (sens de navigation) est mis à INVERSE_1.</td></tr>
<tr><td>referenceTypeId</td><td>NodeId</td><td>Spécifie le NodeId du ReferenceType à suivre. Seules des instances de ce ReferenceType ou de ses sous-types sont retournées.
S'il n'est pas spécifié, toutes les Références sont retournées.</td></tr>
<tr><td>includeSubtypes</td><td>Boolean</td><td>Indique s'il convient d'inclure dans la navigation les sous-types du ReferenceType. Si TRUE, les instances de referenceTypeId et de tous ses sous-types sont retournées.</td></tr>
<tr><td>nodeClassMask</td><td>UInt32</td><td>Spécifie les NodeClasses du TargetNodes. Seuls les TargetNodes avec les NodeClasses sélectionnées sont retournés. Il est affecté aux NodeClasses les bits suivants:
<table>
<tr><th>Bit</th><th>NodeClass</th></tr>
<tr><td>0</td><td>Object</td></tr>
<tr><td>1</td><td>Variable</td></tr>
<tr><td>2</td><td>Method</td></tr>
<tr><td>3</td><td>ObjectType</td></tr>
<tr><td>4</td><td>VariableType</td></tr>
<tr><td>5</td><td>ReferenceType</td></tr>
<tr><td>6</td><td>DataType</td></tr>
<tr><td>7</td><td>View</td></tr>
</table>
S'il est mis à zéro, toutes les NodeClasses sont retournées.</td></tr>
<tr><td>resultMask</td><td>UInt32</td><td>Spécifie les champs dans la structure ReferenceDescription qu'il convient de retourner. Il est affecté aux champs les bits suivants:
<table>
<tr><th>Bit</th><th>Result</th></tr>
<tr><td>0</td><td>ReferenceType</td></tr>
<tr><td>1</td><td>IsForward</td></tr>
<tr><td>2</td><td>NodeClass</td></tr>
<tr><td>3</td><td>BrowseName</td></tr>
<tr><td>4</td><td>DisplayName</td></tr>
<tr><td>5</td><td>TypeDefinition</td></tr>
</table>
Le type ReferenceDescription est défini en 7.24.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>Response</td><td></td><td></td></tr>
<tr><td>responseHeader</td><td>Response Header</td><td>Paramètres communs aux réponses (voir 7.27 pour la définition de ResponseHeader).</td></tr>
</table>

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

results []	BrowseResult	Une liste de *BrowseResults* (résultats de navigation). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du *nodesToBrowse* spécifié dans la demande. Le type *BrowseResult* est défini en 7.3.
diagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les *results* (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de réponse *results*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.8.2.3 Résultats des services

Le Tableau 31 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 31 – Codes de résultats des services Browse

Id symbolique	Description
Bad_ViewIdUnknown	L'identificateur de vue ne renvoie pas à un Noeud de vue valide.
Bad_ViewTimestampInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ViewParameterMismatchInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ViewVersionInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.8.2.4 StatusCodes

Le Tableau 32 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 32 – Codes de résultats de niveau opération de Browse

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceTypeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_BrowseDirectionInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeNotInView	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NoContinuationPoints	Voir le Tableau 166 pour la description de ce code de résultat.
Uncertain_NotAllNodesAvailable	Les résultats de la navigation (Browse) peuvent être incomplets en raison de l'indisponibilité d'un sous-système.

5.8.3 Parcourir le prochain (BrowseNext)

5.8.3.1 Description

Ce *Service* est utilisé pour demander le prochain jeu d'informations de réponse *Browse* ou *BrowseNext* qui sont trop volumineuses pour être envoyées dans une seule réponse. «Trop volumineuses» signifie dans ce contexte que le *Serveur* n'est pas capable de retourner une réponse plus volumineuse ou que le nombre de résultats à retourner dépasse le nombre maximal de résultats à retourner qui avait été spécifié par le *Client* dans la demande Browse d'origine. Le *BrowseNext* doit être présenté dans la même *Session* qui avait été utilisée pour présenter le Browse ou *BrowseNext* qui se poursuit.

5.8.3.2 Paramètres

Le Tableau 33 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 33 – Paramètres du service BrowseNext

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
releaseContinuationPoints	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE les *continuationPoints* passés doivent être réinitialisés pour libérer des ressources dans le *Serveur*. FALSE les *continuationPoints* passés doivent être utilisés pour récupérer le prochain jeu d'informations de navigation. Un *Client* doit toujours utiliser le point de continuation retourné par une réponse *Browse* ou *BrowseNext* pour libérer les ressources pour le point de continuation dans le *Serveur*. Si le *Client* ne souhaite pas récupérer le prochain jeu d'informations de navigation, *BrowseNext* doit être appelé avec ce paramètre mis à TRUE.
continuationPoints []	Continuation Point	Une liste de valeurs opaques définies par le *Serveur* qui représentent des points de continuation. La valeur pour un point de continuation a été retournée au *Client* dans une réponse *Browse* ou *BrowseNext* précédente. Ces valeurs sont utilisées pour identifier la demande *Browse* ou *BrowseNext* précédemment traitée qui se poursuit et le point dans le jeu de résultats à partir duquel la réponse de navigation doit continuer. Les Clients peuvent mélanger les points de continuation provenant de réponses Browse ou BrowseNext différentes. Le type *ContinuationPoint* est décrit en 7.6.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	BrowseResult	Une liste de références qui satisfont aux critères spécifiés dans la demande *Browse* d'origine. La taille et l'ordre de cette liste correspondent à la taille et à l'ordre du paramètre de demande *continuationPoints*. Le type *BrowseResult* est défini en 7.3.
diagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les *results* (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de réponse *results*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.8.3.3 Résultats des services

Le Tableau 34 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 34 – Codes de résultats des services BrowseNext

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.8.3.4 StatusCodes

Le Tableau 35 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 35 – Codes de résultats de niveau opération de BrowseNext

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ReferenceTypeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_BrowseDirectionInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeNotInView	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ContinuationPointInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.8.4 Traduire les chemins en identifiant de nœuds (TranslateBrowsePathsToNodeIds)

5.8.4.1 Description

Ce *Service* est utilisé pour demander au *Serveur* de traduire un ou plusieurs chemins de navigation en des *NodeIds*. Chaque chemin de navigation est composé d'un Noeud de départ et d'un *RelativePath* (chemin relatif). Le *Noeud* de départ spécifié identifie le *Noeud* sur lequel le *RelativePath* est basé. Le *RelativePath* contient une séquence de *ReferenceTypes* et de *BrowseNames*.

Un objectif de ce *Service* est de permettre la programmation en fonction des définitions de type. Sachant que les *BrowseNames* doivent être uniques dans le contexte des définitions de type, un *Client* peut créer un chemin de navigation qui est valide pour une définition de type et utiliser ce chemin sur des instances du type. Par exemple, un *ObjectType* "Boiler" (chaudière) peut avoir une *Variable* "HeatSensor" (capteur de chaleur) comme *InstanceDeclaration*. Un élément graphique programmé en fonction du "Boiler" peut avoir besoin d'afficher la *Value* du "HeatSensor". Si l'élément graphique était appelé "Boiler1", une instance de "Boiler", il aurait besoin d'appeler ce *Service* en spécifiant le *NodeId* de "Boiler1" comme *Nœud* de départ et le *BrowseName* du "HeatSensor" comme chemin de navigation. Le *Service* retournerait le *NodeId* du "HeatSensor" de "Boiler1" et l'élément graphique pourrait s'abonner à son *Attribute Value*.

Si un *Nœud* a plusieurs cibles ayant le même *BrowseName*, le *Serveur* doit retourner une liste de *NodeIds*. Cependant, comme l'un des principaux objectifs de ce *Service* est de prendre en charge la programmation en fonction des définitions de type, le *NodeId* du Nœud basé sur la définition de type du Nœud de départ est retourné comme le premier *NodeId* dans la liste.

5.8.4.2 Paramètres

Le Tableau 36 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 36 – Paramètres du service TranslateBrowsePathsToNodeIds

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
browsePaths []	BrowsePath	Liste des chemins de navigation pour lesquels les *NodeIds* sont demandés.
startingNode	NodeId	*NodeId* du Nœud de départ pour le chemin de navigation.
relativePath	RelativePath	Le chemin à suivre à partir du *startingNode*. Le dernier élément dans le *relativePath* doit toujours avoir un *targetName* spécifié. Cela restreint encore plus la définition du type RelativePath. Le *Serveur* doit retourner *Bad_BrowseNameInvalid* si le *targetName* est absent. La structure *RelativePath* structure est définie en 7.25.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	BrowsePathResult	Liste de résultats pour la liste de chemins de navigation. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *browsePaths*.
statusCode	StatusCode	*StatusCode* pour le chemin de navigation (voir 7.33 pour la définition de *StatusCode*).
targets []	BrowsePathTarget	Liste de cibles pour le *relativePath* à partir du *startingNode*. Un *Serveur* peut rencontrer une *Référence* à un *Nœud* dans un autre *Serveur* qu'il ne peut pas suivre pendant qu'il traite le *RelativePath*. Si cela se produit, le *Serveur* retourne le *NodeId* du Nœud externe et positionne le paramètre *remainingPathIndex* pour indiquer les éléments *RelativePath* qu'il est encore nécessaire de traiter. Pour parachever l'opération, le *Client* doit se connecter à l'autre *Serveur* et appeler de nouveau ce service en utilisant la cible comme le *startingNode* et les éléments non traités comme le *relativePath*.
targetId	ExpandedNodeId	L'identificateur pour une cible du *RelativePath*.
remainingPathIndex	Index	L'indice du premier élément non traité dans le *RelativePath*. Cette valeur doit être égale à la valeur maximale du type de donnée *Index* si tous les éléments ont été traités (voir 7.12 pour la définition d'*Index*).
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour la liste des chemins de navigation (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *browsePaths*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.8.4.3 Résultats des services

Le Tableau 37 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au 7.33.

Tableau 37 – Codes de résultats des services TranslateBrowsePathsToNodeIds

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.8.4.4 StatusCodes

Le Tableau 38 définit les valeurs pour les paramètres *statusCode* de niveau opération qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 38 – Codes de résultats de niveau opération de TranslateBrowsePathsToNodeIds

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat. Ce code indique que le relativePath contenait une liste vide.
Bad_BrowseNameInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Ce code indique qu'un TargetName était absent dans un RelativePath.
Uncertain_ReferenceOutOfServer	L'élément de chemin a des cibles qui sont dans un autre serveur.
Bad_TooManyMatches	L'opération demandée a trop de nombreuses correspondances à retourner. Il convient que les utilisateurs emploient des interrogations pour les volumineux jeux de résultats. Il convient que les *Serveurs* autorisent au moins 10 correspondances avant de retourner ce code d'erreur.
Bad_QueryTooComplex	L'opération demandée exige un trop grand nombre de ressources dans le serveur.
Bad_NoMatch	L'opération n'a aucune correspondance à retourner.

5.8.5 Enregistrer les nœuds (RegisterNodes)

5.8.5.1 Description

Un *Serveur* n'a souvent pas d'accès direct aux informations qu'il gère. Les variables ou les services pourraient se trouver dans des systèmes sous-jacents et un effort supplémentaire est requis pour établir une connexion avec ces systèmes. Le *Service RegisterNodes* peut être utilisé par des *Clients* pour enregistrer les *Noeuds* auxquels ils savent qu'ils vont accéder à répétitions (par exemple Write, Call). Il permet aux *Serveurs* d'installer tout ce qui est nécessaire pour rendre les opérations d'accès plus efficaces. Les *Clients* peuvent espérer des améliorations de performance lorsqu'ils utilisent des *NodeIds* enregistrés, mais les mesures d'optimisation sont spécifiques au vendeur. Pour les *Nodes Variable*, les *Serveurs* doivent concentrer leurs efforts d'optimisation sur l'*Attribut Value*.

La validité des *NodeIds* enregistrés ne peut être garantie que dans la *Session* courante. Les *Clients* doivent immédiatement supprimer les enregistrements des identificateurs inutiles pour libérer des ressources.

RegisterNodes ne valide pas les *NodeIds* issus de la demande. Les *Serveurs* copieront simplement les *NodeIds* inconnus dans la réponse. Les erreurs de structure des *NodeIds* (violations de taille, types d'identificateur non valides) entraîneront la faillite du *Service* complet.

Pour les besoins de l'*Auditing* (Audits), les *Serveurs* ne doivent pas utiliser les *NodeIds* enregistrés, mais ils doivent utiliser seulement les *NodeId*s canoniques qui sont la valeur de l'*Attribut NodeId*.

5.8.5.2 Paramètres

Le Tableau 39 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 39 – Paramètres du service RegisterNodes

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
nodesToRegister []	NodeId	Liste des *NodeIds* à enregistrer que le client a récupérés par navigation, interrogation ou de quelque autre manière.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
registeredNodeIds []	NodeId	Une liste de *NodeIds* que le *Client* doit utiliser pour des opérations d'accès ultérieures. La taille et l'ordre de cette liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToRegister*. Le *Serveur* peut retourner le *NodeId* de la demande ou un nouveau *NodeId* (un pseudonyme). Il est recommandé que le *Serveur* retourne un *NodeIds* numérique pour la dénomination. Si aucune optimisation n'est prise en charge pour un Noeud, le *Serveur* doit retourner le *NodeId* issu de la demande.

5.8.5.3 Résultats des services

Le Tableau 40 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 40 – Codes de résultats des services RegisterNodes

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Les *Serveurs* doivent rejeter complètement la demande *RegisterNodes* si l'un quelconque des *NodeIds* dans le paramètre *nodesToRegister* est structurellement non valide.

5.8.6 Supprimer l'enregistrement de nœuds (UnregisterNodes)

5.8.6.1 Description

Ce *Service* est utilisé pour supprimer l'enregistrement des *NodeIds* qui ont été obtenus via le service *RegisterNodes*.

UnregisterNodes ne valide pas les *NodeIds* issus de la demande. Les *Serveurs* doivent simplement supprimer l'enregistrement des *NodeIds* qui sont connus comme des *NodeIds* enregistrés. Tous les *NodeIds* qui sont dans la liste, mais ne sont pas des *NodeIds* enregistrés, sont simplement ignorés.

5.8.6.2 Paramètres

Le Tableau 41 définit les paramètres du *Service*.

Tableau 41 – Paramètres du service UnregisterNodes

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
nodesToUnregister []	NodeId	Une liste des *NodeIds* qui ont été obtenus via le service *RegisterNodes*.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.8.6.3 Résultats des services

Le Tableau 42 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 42 – Codes de résultats des services UnregisterNodes

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.9 Jeu de service d'interrogation (Query Service Set)

5.9.1 Vue générale

Ce *Service Set* est utilisé pour émettre une *Query* (interrogation) vers un *Serveur*. La *Query* OPC UA est générique en ce qu'elle fournit un mécanisme de stockage sous-jacent indépendant de la fonction *Query* qui peut être utilisée pour accéder à une grande variété d'unités de stockage de données et de systèmes de gestion d'informations d'OPC UA. La *Query* OPC UA permet à un *Client* d'accéder à des données maintenues par un *Serveur* sans connaître le plan logique utilisé pour le stockage interne des données. La connaissance de l'Espace d'Adresses est suffisante.

Une *Application OPC UA* est censée utiliser les *Services Query* d'OPC UA comme une partie d'un processus d'initialisation ou une étape occasionnelle de synchronisation d'informations. Par exemple, *Query* OPC UA serait utilisée pour un accès à des données en vrac d'une unité de stockage persistante pour initialiser une application d'analyse avec l'état courant d'une configuration système. Une *Query* peut aussi être utilisée pour initialiser ou alimenter des données en vue d'un rapport.

Une *Query* définit les instances d'un ou plusieurs *TypeDefinitionNodes* dans l'Espace d'Adresses qu'il convient d'appliquer à un jeu d'*Attributs*. Les résultats retournés par un *Serveur* sont sous la forme d'une matrice de *QueryDataSets* (jeu de données d'interrogation). Les valeurs d'*Attributs* sélectionnées dans chaque *QueryDataSet* proviennent de la définition des *TypeDefinitionNodes ou TypeDefinitionNodes connexes* sélectionnés et les résultats apparaissent dans le même ordre que les *Attributs* qui ont été passés dans la *Query*. La *Query* prend également en charge le filtrage des Noeuds en fonction des valeurs d'*Attributs*, ainsi que les relations entre *TypeDefinitionNodes*.

Voir Annexe B pour des exemples d'interrogations.

5.9.2 Interroger des vues (Querying Views)

Une Vue est un sous-ensemble de l'Espace d'Adresses disponible dans le *Serveur*. Voir la CEI 62541-5 pour une description de l'organisation des Vues dans l'Espace d'Adresses.

Pour toute Vue existante, une *Query* peut être utilisée pour retourner un sous-ensemble de données de la Vue. Lorsqu'une application émet une *Query* en fonction d'une Vue, seules les données définies par la Vue sont retournées. Les données non incluses dans la Vue mais incluses dans l'Espace d'Adresses d'origine ne sont pas retournées.

Les *Services Query* prennent en charge l'accès à des données courantes et historiques. Le *Service* prend en charge un *Client* qui interroge une version antérieure de l'Espace d'Adresses. Les Clients peuvent spécifier une *ViewVersion* (Version de vue) ou un *Timestamp* (Marqueur temporel) dans une *Query* pour accéder à des versions antérieures de l'Espace d'Adresses. Le service Query OPC UA est complémentaire au service Historical Access (Accès à historique) en ce que le premier est utilisé pour *Query* (interroger) un *Espace d'Adresses* qui existait à un moment donné et que le second est utilisé pour *Query* afin d'obtenir la valeur des *Attributs* au fil du temps. Ainsi, une *Query* peut être utilisée pour récupérer une partie d'un *Espace d'Adresse* antérieur et, de ce fait, un historique des valeurs

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

de l'*Attribut* sera accessible à l'aide de Historical Access même si le *Noeud* ne se trouve plus dans *l'Espace d'Adresses* courant.

Les *Serveurs* qui prennent en charge *Query* sont censés être capables d'accéder à *l'Espace d'Adresses* qui est associé au *Serveur* local et à n'importe quelles *Vues* qui sont disponibles sur le *Serveur* local. Si une *Vue* ou l'espace adresse référence également un *Serveur* distant, l'interrogation peut être capable d'accéder à l'espace adresse du *Serveur* distant, mais cela n'est pas obligatoire. Si un *Serveur* accède effectivement à un *Serveur* distant, l'accès doit être accompli en utilisant l'identité d'utilisateur du *Client* tel que décrit en 5.5.1.

5.9.3 Emettre une demande (QueryFirst)

5.9.3.1 Description

Ce *Service Set* est utilisé pour émettre une demande *Query* vers le *Serveur*. La complexité de la *Query* peut etre très simple ou hautement sophistiquée. La *Query* peut simplement demander des données issues d'instances d'un *TypeDefinitionNode* ou *TypeDefinitionNode* soumis à des restrictions spécifiées par le filtre. D'autre part, la *Query* peut demander des données issues d'instances de type Noeud*s* connexes en spécifiant un *RelativePath* partant d'un *TypeDefinitionNode* d'origine. Dans le filtre, un jeu séparé de chemins peut être construit pour limiter les instances qui fournissent les données. Un chemin filtrant peut inclure plusieurs opérateurs *RelatedTo* (Relié à) pour définir un chemin à sauts multiples entre des instances sources et des instances cibles. Par exemple, on pourrait passer au filtre des étudiants d'une école particulière, mais retourner des informations relatives aux étudiants et à leurs familles. Dans ce cas, la relation étudiant-école est parcourue pour le filtrage, mais la relation étudiant-famille est parcourue pour sélectionner des données. Pour une description complète de *ContentFilter,* voir 7.4; voir aussi l'Article B.1 pour des exemples simples et l'Article B.2 pour de plus complexes exemples de filtre de contenu et d'interrogations.

Le *Client* fournit une matrice de *NodeTypeDescriptions* qui spécifient le *NodeId* d'un *TypeDefinitionNode* et sélectionne les *Attributs* qui doivent être retournés dans la réponse. Un client peut aussi fournir un jeu de *RelativePaths* à travers le système de type commençant à partir d'un *TypeDefinitionNode* de départ. En utilisant ces chemins, le client sélectionne un jeu d'*Attributs* à partir *des Noeuds* qui sont reliés à des instances du *TypeDefinitionNode* de départ. En outre, le *Client* peut demander au *Serveur* de retourner des instances des sous-types de *TypeDefinitionNodes*. Si un *Attribut* sélectionné n'existe pas dans un *TypeDefinitionNode* mais existe dans un sous-type, il est admis avoir une valeur null dans le *TypeDefinitionNode* en question. Par conséquent, ceci ne constitue pas une condition d'erreur et une valeur null est retournée pour l'*Attribut*.

Le *Client* peut utiliser le paramètre filtre pour limiter le jeu de résultats en limitant les *Attributs* et les *Properties* à certaines valeurs. Une autre manière dont le *Client* peut utiliser un filtre pour limiter le jeu de résultats consiste à spécifier comment il convient de relier les instances, en utilisant des opérateurs *RelatedTo*. Dans ce cas, si une instance au sommet du chemin *RelatedTo* ne peut pas être suivie jusqu'au bas du chemin via des sauts spécifiés, aucun *QueryDataSet* n'est retourné pour l'instance de départ ou l'une quelconque des instances intermédiaires.

Lorsqu'il effectue une interrogation relative à des instances connexes dans le *RelativePath*, le *Client* peut facultativement demander des *Références*. Une *Référence* est demandée via un RelativePath qui inclut seulement un *ReferenceType*. Si toutes les *Références* sont désirées, le *ReferenceType* racine serait énumérée. Ces *Références* sont retournées comme partie des *QueryDataSets*.

5.9.3.2 Paramètres

Le Tableau 43 définit les paramètres de demande et le Tableau 44 les paramètres de réponse pour le *Service QueryFirst*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 43 – Paramètres de demande QueryFirst

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
view	ViewDescription	Spécifie une Vue et un contexte temporel au *Serveur* (voir 7.37 pour la description de *ViewDescription*).
nodeTypes []	NodeTypeDescription	Il s'agit de la description du type de *Noeud*.
typeDefinitionNode	ExpandedNodeId	*NodeId* du *TypeDefinitionNode* de départ des instances pour lesquelles les données sont à retourner.
includeSubtypes	Boolean	Un fanion qui indique si, oui ou non, il convient que le *Serveur* inclue des instances des sous-types de TypeDefinitionNode dans la liste des instances du type *Node*.
dataToReturn []	QueryDataDescription	Spécifie un *Attribut* ou une *Référence* partant du typeDefinitionNode de départ le long d'un relativePath donné pour lequel retourner des données.
relativePath	RelativePath	Chemin de navigation relatif au Noeud de départ qui identifie le Noeud contenant les données demandées, le Noeud de départ étant un Noeud d'instance du type défini le Noeud de définition de type. Les Noeuds d'instance sont l'objet d'une limitation supplémentaire par le filtre fourni comme partie de cet appel. Pour une définition de relativePath, voir 7.25. Ce chemin relatif pourrait se terminer sur une *Référence* et, dans ce cas, la *ReferenceDescription* de la *Référence* serait retournée comme étant sa valeur.
attributeID	IntegerId	Identificateur de l'*Attribut*. Il doit être un identificateur d'*Attribut*valide. L'*IntegerId* est défini en 7.13. Les IntegerIds pour les attributs sont définis dans la CEI 62541-6. Si le *RelativePath* s'est terminé à une *Référence*, ce paramètre est 0 et ignoré par le serveur.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une structure ou d'une matrice ou bien une plage simple d'indices pour les matrices. Si une plage d'éléments est spécifiée, les valeurs sont retournées sous forme de composé. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini en 7.21. Ce paramètre est null si l'*Attribut* spécifié n'est pas une matrice ou une structure. Cependant, si l'*Attribut* spécifié est une matrice ou une structure, et ce paramètre est donc null, tous les éléments doivent être inclus dans la plage.
filter	ContentFilter	Les Noeuds obtenus doivent être limités aux Noeuds qui correspondent aux critères définis par le filtre. ContentFilter est l'objet d'un débat en 7.4. Si un filtre vide est fourni, l'espace adresse tout entier doit être examiné et tous les Noeuds contenant un attribut ou une *Référence* demandé(e) correspondant aux critères sont retournés.
maxDataSetsToReturn	Counter	Le nombre de *QueryDataSets* que le *Client* souhaite que le *Serveur* retourne dans la réponse et sur chaque réponse ultérieure d'appel de continuation. Le *Serveur* est autorisé à limiter encore plus la réponse, mais ne doit pas dépasser cette limite. Une valeur 0 indique que le *Client* n'impose aucune limitation.
maxReferencesToReturn	Counter	Le nombre de *Références* que le *Client* souhaite que le *Serveur* retourne dans la réponse pour chaque *QueryDataSet* et sur chaque réponse ultérieure d'appel de continuation. Le *Serveur* est autorisé à limiter encore plus la réponse, mais ne doit pas dépasser cette limite. Une valeur 0 indique que le *Client* n'impose aucune limitation. Par exemple, dans le cas d'un résultat où quatre Noeuds sont retournés, mais chacun a 100 *Références* et cette limite est fixée à 50, seules les 50 premières *Références* pour chaque Noeud seront retournées sur l'appel initial et un point de continuation sera établi pour indiquer l'existence de données supplémentaires.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 44 – Paramètres de réponse QueryFirst

Nom	Type	Description
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
queryDataSets []	QueryDataSet	La matrice de *QueryDataSets*. Cette matrice est vide s'il n'y a pas des Noeuds ou de *Références* ayant satisfait aux critères de *nodeTypes*. Dans ce cas, le paramètre continuationPoint doit être vide. Le type *QueryDataSet* est défini en 7.22.
continuationPoint	ContinuationPoint	Valeur opaque définie par le serveur qui identifie le point de continuation. Le point de continuation est utilisé seulement lorsque les résultats de *Query* sont trop volumineux pour être retournés dans une seule réponse. «Trop volumineux» signifie dans ce contexte que le *Serveur* n'est pas capable de retourner une réponse plus volumineuse ou que le nombre de *QueryDataSets* à retourner dépasse le nombre maximal de *QueryDataSets* à retourner qui avait été spécifié par le *Client* dans la demande. Le point de continuation est utilisé dans le *Service QueryNext*. Lorsqu'il n'est pas utilisé, la valeur de ce paramètre est null. Si un point de continuation est retourné, le *Client* doit appeler *QueryNext* pour récupérer le prochain jeu de *QueryDataSets* ou pour libérer des ressources pour le point de continuation dans le *Serveur*. Un point de continuation doit rester actif jusqu'à ce que le *Client* passe le point de continuation à *QueryNext* ou jusqu'à ce que la session soit fermée. Si le nombre maximum de points de continuation a été atteint, le point de continuation le plus ancien doit être réinitialisé. Le type *ContinuationPoint* est décrit en 7.6.
parsingResults []	ParsingResult	Liste de résultats d'analyse pour *QueryFirst*. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *NodeTypes*. Cette liste est garnie de tous les codes de statut qui se rapportent au traitement des types de nœuds qui font partie de l'interrogation. La matrice peut être vide s'il ne s'est pas produit d'erreurs. Si un quelconque type de nœud a rencontré une erreur, tous les types de nœuds doivent avoir un code de statut associé.
statusCode	StatusCode	Résultat d'analyse pour la *NodeTypeDescription* demandée.
dataStatusCodes []	StatusCode	Liste de résultats pour *dataToReturn*. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *dataToReturn*. La matrice peut être vide s'il ne s'est pas produit d'erreurs.
dataDiagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *dataToReturn* (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *dataToReturn*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de l'interrogation.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour la *NodeTypeDescription* demandée. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande d'interrogation.
filterResult	ContentFilter Résultat	Une structure qui contient toutes les éventuelles erreurs associées au filtre. Cette structure doit être vide s'il ne s'est pas produit d'erreurs. Le type *ContentFilterResult* est défini en 7.4.2.

5.9.3.3 Résultats des services

Si la *Query* est invalide ou ne peut pas être traitée, les *QueryDataSets* ne sont pas retournés et seul est retourné un *Service* result, filterResult, parsingResults et *DiagnosticInfos* facultatives. Le Tableau 45 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 45 – Codes de résultats des services QueryFirst

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ContentFilterInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_ViewIdUnknown	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ViewTimestampInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ViewParameterMismatchInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ViewVersionInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_InvalidFilter	Le filtre fourni n'est pas valide, voir le filterResults pour les erreurs spécifiques.
Bad_NodelistError	Les NodeTypes fournis contiennent une erreur, voir le parsingResults pour les erreurs spécifiques.
Bad_InvalidView	La description de vue fournie n'est pas une description de Vue valide.
Good_ResultsMayBeIncomplete	Il convient que le serveur ait suivi une référence à un nœud dans un serveur distant, mais il ne l'a pas fait. Le jeu de résultats peut être incomplet.

5.9.3.4 StatusCodes

Le Tableau 46 définit les valeurs pour le paramètre *statusCode* de parsingResults qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 46 – Codes de résultats de niveau opération de QueryFirst

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NotTypeDefinition	Le Nodeid fourni n'était pas un nodeid de définition de type.
Bad_AttribteIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.9.4 Poursuivre la demande (QueryNext)

5.9.4.1 Descriptions

Ce *Service* est utilisé pour demander le prochain jeu d'informations de réponse *QueryFirst* ou *QueryNext* qui sont trop volumineuses pour être envoyées dans une seule réponse. «Trop volumineuses» signifie dans ce contexte que le *Serveur* n'est pas capable de retourner une réponse plus grosse ou que le nombre de *QueryDataSets* à retourner dépasse le nombre maximal de *QueryDataSets* à retourner qui avait été spécifié par le *Client* dans la demande d'origine. Le *QueryNext* doit être présenté dans la même session qui avait été utilisée pour présenter le *QueryFirst* ou *QueryNext* qui se poursuit.

5.9.4.2 Paramètres

Le Tableau 47 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 47 – Paramètres du service QueryNext

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
releaseContinuationPoint	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE le *continuationPoint* passé doit être réinitialisé pour libérer des ressources pour le point de continuation dans le *Serveur*. FALSE le *continuationPoint* passé doit être utilisé pour récupérer le prochain jeu de *QueryDatSets*. Un *Client* doit toujours utiliser le point de continuation retourné par une réponse *QueryFirst* ou *QueryNext* pour libérer les ressources pour le point de continuation dans le *Serveur*. Si le *Client* ne souhaite pas récupérer le prochain jeu d'informations de *Query*, *QueryNext* doit être appelé avec ce paramètre mis à TRUE. Si le paramètre est mis à TRUE, tous les paramètres de matrices dans la réponse doivent contenir des matrices vides.
continuationPoint	ContinuationPoint	Valeur opaque définie par le serveur qui représente le point de continuation. La valeur pour le point de continuation a été retournée au *Client* dans une réponse *QueryFirst* ou *QueryNext* précédente. Cette valeur est utilisée pour identifier la demande *QueryFirst* ou *QueryNext* précédemment traitée qui se poursuit et le point dans le jeu de résultats à partir duquel la réponse de navigation doit continuer. Le type *ContinuationPoint* est décrit en 7.6.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
queryDataSets []	QueryDataSet	La matrice de *QueryDataSets*. Le type *QueryDataSet* est défini en 7.22.
revisedContinuationPoint	ContinuationPoint	Valeur opaque définie par le serveur qui représente le point de continuation. Elle est utilisée seulement si les informations à retourner sont trop volumineuses pour être contenues dans une seule réponse. Lorsqu'elle n'est pas utilisée ou lorsque le *releaseContinuationPoint* est mis, la valeur de ce paramètre est null. Le type *ContinuationPoint* est décrit en 7.6.

5.9.4.3 Résultats des services

Le Tableau 48 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 48 – Codes de résultats des services QueryNext

Id symbolique	Description
Bad_ContinuationPointInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.10 Jeu de services pour les attributs (Attribute Service Set)

5.10.1 Vue générale

Ce *Service Set* fournit les *Services* pour accéder à des *Attributs* qui sont partie intégrante des Noeuds.

5.10.2 Lecture (Read)

5.10.2.1 Description

Ce *Service* est utilisé pour lire un ou plusieurs *Attributs* d'un ou plusieurs Noeuds. Pour les valeurs d'*Attributs* construits dont les éléments ont des indices, comme une matrice, ce *Service* permet aux *Clients* de lire la totalité du jeu de valeurs indicées comme un composé, de lire des éléments individuels ou de lire des plages d'éléments du composé.

Le paramètre maxAge est utilisé pour ordonner au *Serveur* d'accéder à la valeur à partir de la source de données sous-jacente (telle qu'un dispositif par exemple) si sa copie des données

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

est plus ancienne que celle que le maxAge spécifie. Si le *Serveur* ne peut pas satisfaire à l'âge maximal demandé, il retourne la valeur la plus appropriée au lieu de rejeter la demande.

5.10.2.2 Paramètres

Le Tableau 49 définit les paramètres du *Service*.

Tableau 49 – Paramètres du service Read

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
maxAge	Duration	Âge maximal de la valeur à lire en millisecondes. L' âge de la valeur est basé sur la différence entre le *ServeurTimestamp* (horodatage du serveur) et l'heure à laquelle le *Serveur* commence à traiter la demande. Par exemple, si le *Client* spécifie un *maxAge* de 500 ms et la durée d'attente est de 100 millisecondes avant que le *Serveur* ne commence à traiter la demande, l'âge de la valeur retournée pourrait être 600 ms avant l'heure à laquelle elle a été demandée. Si le *Serveur* a une ou plusieurs valeurs d'un *Attribut* qui sont dans la limite de l'âge maximal, il peut retourner n'importe laquelle de ces valeurs ou il peut lire une nouvelle valeur dans la source de données. Le nombre de valeurs d'un *Attribut* que possède un *Serveur* dépend du nombre de *MonitoredItems* qui sont définis pour cet *Attribut*. Dans tous les cas, le *Client* ne peut présumer de la copie des données qui sera retournée. Si le *Serveur* n'a pas de valeur s'inscrivant dans la limite de l'âge maximal, il doit tenter de lire une nouvelle valeur dans la source de données. Si le *Serveur* ne peut pas satisfaire au *maxAge* demandé, il retourne sa valeur de "meilleur effort" au lieu de rejeter la demande. Ce cas de figure peut se produire lorsque le temps mis par le *Serveur* pour traiter et retourner la nouvelle valeur de donnée après accès est supérieur à l'âge maximal spécifié. Si le *maxAge* est mis à 0, le *Serveur* doit tenter de lire une nouvelle valeur dans la source de données. Si *maxAge* est mis à une valeur supérieure ou égale à la valeur Int32 maximale, le *Serveur* doit tenter de récupérer une valeur mise en cache. Les valeurs négatives ne sont pas valides pour *maxAge*.
timestampsTo Return	enum TimestampsTo Return	Une énumération qui spécifie les *Timestamps* (Marqueurs temporels) à retourner pour chaque *Variable Value Attribute*. L'énumération *TimestampsToReturn* est définie en 7.34.
nodesToRead []	ReadValueId	Liste des Noeuds et de leurs *Attributs* à lire. Pour chaque entrée dans la liste, un *StatusCode* est retourné, et s'il indique un succès, l'*Attribute Value* est également retourné. Le type de paramètre ReadValueId est défini en 7.23 .
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	DataValue	Liste de valeurs d'*Attribut* (voir 7.7 pour la définition *DataValue*). La taille et l'ordre de cette liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToRead*. Il y a une entrée dans cette liste pour chaque Noeud contenu dans le paramètre *nodesToRead*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de cette liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToRead*. Il y a une entrée dans cette liste pour chaque Noeud contenu dans le paramètre *nodesToRead*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.10.2.3 Résultats des services

Le Tableau 50 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 50 – Codes de résultats des services Read

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MaxAgeInvalid	Le paramètre âge maximal n'est pas valide.
Bad_TimestampsToReturnInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.10.2.4 StatusCodes

Le Tableau 51 définit les valeurs pour le *statusCode* de niveau opération contenu dans la structure *DataValue* de chaque élément *values*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 51 – Codes de résultats de niveau opération de Read

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_AttributeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeNoData	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NotReadable	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.

5.10.3 Lecture de l'historique (HistoryRead)

5.10.3.1 Description

Ce *Service* est utilisé pour lire des valeurs ou *Evénements* historiques d'un ou plusieurs Noeuds. Pour les valeurs d'*Attributs* construits dont les éléments ont des indices, comme une matrice, ce *Service* permet aux *Clients* de lire la totalité du jeu de valeurs indicées comme un composé, à lire des éléments individuels ou de lire des plages d'éléments du composé. Les *Serveurs* peuvent rendre les valeurs historiques disponibles aux *Clients* en utilisant ce *Service*, bien que les valeurs historiques elles-mêmes ne soient pas visibles dans *l'Espace d'Adresses*.

L'*Attribute AccessLevel* (Attribut Niveau d'accès) défini dans la CEI 62541-3 indique la prise en charge des valeurs historiques par des Noeuds. Plusieurs paramètres de demande indiquent comment le *Serveur* doit accéder à des valeurs de la source sous-jacente des données de l'historique. L'*Attribut EventNotifier* (Attribut Notificateur d'événement) défini dans la CEI 62541-3 indique la prise en charge d'*Evénements* historiques par des Noeuds.

Le paramètre *continuationPoint* dans l'*HistoryRead* est utilisé pour marquer le point à partir duquel doit se poursuivre la lecture si toutes les valeurs n'ont pas pu être retournées dans une seule réponse. La valeur n'est pas connue du *Client* et elle est utilisée seulement pour maintenir les informations d'état pour le point de continuation pour le *Serveur*. Un *Serveur* peut utiliser le marqueur temporel du dernier élément de donnée retourné si le marqueur temporel est unique. Cela réduit la nécessité de stocker des informations d'état dans le *Serveur* pour le point de continuation.

Pour des détails supplémentaires relatifs à la lecture de données historiques et d'*Evénements* historiques, voir la CEI 62541-11.

5.10.3.2 Paramètres

Le Tableau 52 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 52 – Paramètres du service HistoryRead

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
historyReadDetails	Extensible Parameter HistoryReadDetails	Les détails définissent les types de lecteurs d'historiques qui peuvent être effectuées. Le type de paramètre *HistoryReadDetails* est un type de paramètre extensible qui est défini formellement dans la CEI 62541-11. Le type *ExtensibleParameter* est défini en 7.11.
timestampsToReturn	enum TimestampsTo Return	Une énumération qui spécifie les marqueurs temporels à retourner pour chaque *Variable Value Attribute*. L'énumération *TimestampsToReturn* est définie en 7.34. Il n'est pas valide de spécifier un *TimestampsToReturn* de NEITHER. Dans ce cas, le *Server* doit retourner un *StatusCode Bad_InvalidTimestampArgument*.
releaseContinuation Points	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE les *continuationPoints* passés doivent être réinitialisés pour libérer des ressources dans le *Serveur*. FALSE les *continuationPoints* passés doivent être utilisés pour récupérer le prochain jeu d'informations historiques. Un *Client* doit toujours utiliser le point de continuation retourné par une réponse *HistoryRead* pour libérer les ressources pour le point de continuation dans le *Serveur*. Si le *Client* ne souhaite pas récupérer le prochain jeu d'informations historiques, *HistoryRead* doit être appelé avec ce paramètre mis à TRUE.
nodesToRead []	HistoryReadValueId	Ce paramètre contient la liste d'éléments sur lesquels la récupération d'historique est à accomplir.
nodeId	NodeId	Si l'*HistoryReadDetails* est RAW, PROCESSED, MODIFIED or ATTIME: Le *nodeId* des Noeuds dont les valeurs historiques sont à lire. La valeur retournée doit toujours comprendre un marqueur temporel. Si l'*HistoryReadDetails* est EVENTS: Le *NodeId* du Noeud dont l'historique des *Evénements* doit être lu. Si le Noeud ne prend pas en charge l'accès demandé pour des valeurs historiques ou des *Evénements* historiques, une réponse d'erreur appropriée pour le Noeud donné doit être générée.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices. Si une plage d'éléments est spécifiée, les valeurs sont retournées sous forme de composé. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini en 7.21. Ce paramètre est null si la valeur n'est pas une matrice. Cependant, si la valeur est une matrice, et ce paramètre est donc null, tous les éléments doivent être inclus dans la plage.
dataEncoding	QualifiedName	Un *QualifiedName* qui spécifie le codage de donnés à retourner pour la *Value* à lire (voir 7.23 pour la définition de la façon de spécifier le codage de données). Le paramètre est ignoré lors de la lecture de l'historique d'*Evénements*.
continuationPoint	ByteString	Pour chaque *NodeToRead*, ce paramètre spécifie un point de continuation retourné à partir d'un précédent appel *HistoryRead*, permettant au *Client* de poursuivre cette lecture à partir de la dernière valeur reçue. L'*HistoryRead* est utilisé pour sélectionner une séquence ordonnée de valeurs ou événements historiques. Un point de continuation marque un point dans cette séquence ordonnée et, ainsi, le *Serveur* retourne le sous-ensemble de la séquence qui suit le point en question. Une valeur null indique que ce paramètre n'est pas utilisé. Ce point de continuation est décrit en plus de détails dans la CEI 62541-11.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour le type *ResponseHeader*).
results []	HistoryReadResult	Liste de résultats lus. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToRead*.
statusCode	StatusCode	*StatusCode* pour le *NodeToRead* (voir 7.33 pour la définition de *StatusCode*).
continuationPoint	ByteString	Ce paramètre est utilisé seulement si le nombre des valeurs à retourner est trop grand pour que celles-ci soient retournées dans une seule réponse. Lorsque ce paramètre n'est pas utilisé, sa valeur est null. Les *Serveurs* doivent prendre en charge au moins un point de continuation par *Session*. Les *Serveurs* spécifient un nombre maximal de points de continuation d'historiques par *Session* dans l'*Object* fonctions du *Serveur* défini dans la CEI 62541-5. Un point de continuation doit rester actif jusqu'à ce que le *Client* passe le point de continuation à *HistoryRead* ou jusqu'à ce que la *Session* soit fermée. Si le nombre maximal de points de continuation a été atteint, le point de continuation le plus ancien doit être réinitialisé.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
Request		
historyData	Extensible Parameter HistoryData	Les données d'historique retournées par le Noeud. Le type de paramètre *HistoryData* est un type de paramètre extensible qui est défini formellement dans la CEI 62541-11. Il spécifie les types des données d'historique qui peuvent être retournées. Le type de base *ExtensibleParameter* est défini en 7.11.
diagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToRead*. Il y a une entrée dans cette liste pour chaque Noeud contenu dans le paramètre *nodesToRead*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.10.3.3 Résultats des services

Le Tableau 53 définit les résultats de *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 53 – Codes de résultats des services HistoryRead

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TimestampsToReturnInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_HistoryOperationInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_HistoryOperationUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.

5.10.3.4 StatusCodes

Le Tableau 54 définit les valeurs pour le paramètre *statusCode* de niveau opération qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166. Les *StatusCodes* spécifiques à l'accès aux historiques sont définis dans la CEI 62541-11.

Tableau 54 – Codes de résultats de niveau opération de HistoryRead

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_ContinuationPointInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_InvalidTimestampArgument	Le marqueur temporel défini à retourner n'était pas valide.

5.10.4 Ecrire (Write)

5.10.4.1 Description

Ce *Service* est utilisé pour écrire des valeurs dans un ou plusieurs *Attributs* d'un ou plusieurs Noeuds. Pour les valeurs d'*Attributs* construits dont les éléments ont des indices, comme une matrice, ce *Service* permet aux *Clients* d'écrire la totalité du jeu de valeurs à indices comme un composé, d'écrire des éléments individuels ou d'écrire des plages d'éléments du composé.

Les valeurs sont écrites dans la source de données (telle qu'un dispositif par exemple) et le *Service* ne retourne rien jusqu'à ce qu'il écrive les valeurs ou détermine que la valeur ne peut pas être écrite. Dans certains cas, le *Serveur* peut écrire avec succès dans un système ou *Serveur* intermédiaire et ne saura pas si la source de données avait été correctement mise à jour. Dans ces cas, il convient que le *Serveur* rapporte un code de succès qui indique que l'écriture n'a pas été vérifiée. Dans les cas où le *Serveur* est à même de vérifier qu'il a réussi à écrire dans la source de données, il rapporte un succès inconditionnel.

Il est possible que le *Serveur* puisse réussir à écrire certains *Attributs* et pas d'autres. Le retour en arrière est du ressort du *Client*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Si un *Serveur* autorise l'écriture d'*Attributs* ayant le *DataType* LocalizedText, le Client peut ajouter ou écraser le texte pour un paramètre de lieu en écrivant le texte avec le *LocaleId* associé. Le fait d'écrire une *String* (Chaîne) null pour un paramètre de lieu doit supprimer la *String* pour le paramètre de lieu en question. Le fait d'écrire une *String* null pour le texte et une *String* null pour le *LocaleId* doit supprimer les entrées pour tous les paramètres de lieu.

5.10.4.2 Paramètres

Le Tableau 55 définit les paramètres du *Service*.

Tableau 55 – Paramètres du service Write

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
nodesToWrite []	WriteValue	Liste des Noeuds et de leurs *Attributs* à écrire.
nodeId	NodeId	*NodeId* du Noeud qui contient les *Attributs*.
AttributeID	IntegerId	Identificateur de l'*Attribut*. Il doit être un identificateur d'*Attribut* valide. L'*IntegerId* est défini en 7.13. Les IntegerIds pour les attributs sont définis dans la CEI 62541-6.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini en 7.21. Ce paramètre n'est pas utilisé si l'*Attribut* spécifié n'est pas une matrice. Cependant, si l'*Attribut* spécifié est une matrice et ce paramètre n'est pas utilisé, tous les éléments doivent être inclus dans la plage. Ce paramètre est null s'il n'est pas utilisé.
value	DataValue	La valeur d'*Attribut* du Noeud (voir 7.7 pour la définition de *DataValue*). Si le paramètre *indexRange* est spécifié, la *Value* doit être une matrice même si un seul élément est en cours d'écriture. Si le *SourceTimestamp* ou le *ServeurTimestamp* est spécifié, le *Serveur* doit utiliser ces valeurs. Le *Serveur* retourne une erreur Bad_WriteNotSupported s'il ne prend pas en charge l'écriture de marqueurs temporels. Un *Serveur* doit retourner une erreur Bad_TypeMismatch si le type de données de la valeur n'est pas identique au datatype, ou un sous-type de celui-ci, pour l'*Attribut*. Le *Serveur* retourne une erreur Bad_DataEncodingUnsupported s'il ne prend pas en charge l'écriture du codage de données passé.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des résultats pour les Noeuds à écrire (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToWrite*. Il y a une entrée dans cette liste pour chaque Noeud contenu dans le paramètre *nodesToWrite*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les Noeuds à écrire (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *nodesToWrite*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.10.4.3 Résultats des services

Le Tableau 56 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 56 – Codes de résultats des services Write

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.10.4.4 StatusCodes

Le Tableau 57 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 57 – Codes de résultats de niveau opération de Write

Id symbolique	Description
Good_CompletesAsynchronously	Voir le Tableau 165 pour la description de ce code de résultat. La valeur a été écrite avec succès dans un système intermédiaire mais le *Serveur* ne sait pas si la source de données avait été correctement mise à jour.
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_AttributeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeNoData	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_WriteNotSupported	L'opération d'écriture demandée n'est pas prise en charge. Si un *Client* tente d'écrire n'importe quelle combinaison de "value", "quality", "timestamp" alors que le *Serveur* ne prend pas en charge la combinaison demandée (qui pourrait être une seule grandeur telle que juste un marqueur temporel), le *Serveur* ne doit pas effectuer d'écriture sur ce Noeud et doit retourner ce *StatusCode* pour ce Noeud.
Bad_NotWritable	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat. L'utilisateur courant n'a pas la permission d'écrire l'attribut.
Bad_OutOfRange	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_TypeMismatch	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.

5.10.5 Mise à jour de l'historique (HistoryUpdate)

5.10.5.1 Description

Ce *Service* est utilisé pour mettre à jour des valeurs ou *Evénements* historiques d'un ou plusieurs Noeuds. Plusieurs paramètres de demande indiquent comment le *Serveur* doit mettre à jour une valeur ou un *Evénement* historique. Les actions valides sont Insert (Insérer), Replace (Remplacer) ou Delete (Supprimer).

5.10.5.2 Paramètres

Le Tableau 58 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 58 – Paramètres du service HistoryUpdate

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
historyUpdateDetails []	Extensible Parameter HistoryUpdate Details	Les détails définis pour la mise à jour. Le type de paramètre *HistoryUpdateDetails* est un type de paramètre extensible qui est défini formellement dans la CEI 62541-11. Il spécifie les types des mises à jour d'historique qui peuvent être accomplies. Le type *ExtensibleParameter* est défini en 7.11.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	HistoryUpdate Result	Liste des résultats de mise à jour pour les détails de mise à jour d'historiques. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments de détails du paramètre *historyUpdateDetails* spécifié dans la demande.
statusCode	StatusCode	*StatusCode* pour la mise à jour du Noeud (voir 7.33 pour la définition de *StatusCode*).
operationResults []	StatusCode	Liste des *StatusCodes* pour les opérations à exécuter sur un Noeud. La taille et l'ordre de la liste correspondent à la taille et à l'ordre de toute liste définie par l'élément "details" communiqué dans un rapport par cette entrée de *updateResults*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les opérations à exécuter sur un Noeud (voir 7.8 pour la définition de *DiagnosticInfo)*. La taille et l'ordre de la liste correspondent à la taille et à l'ordre de toute liste définie par l'élément "details" communiqué dans un rapport par cette entrée de *updateResults*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les détails de mise à jour d'historiques. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments de détails du paramètre *historyUpdateDetails* spécifié dans la demande. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.10.5.3 Résultats des services

Le Tableau 59 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 59 – Codes de résultats des services HistoryUpdate

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.10.5.4 StatusCodes

Le Tableau 60 définit les valeurs pour les paramètres *statusCode* et *operationResult* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166. Les *StatusCodes* spécifiques à l'accès aux historiques sont définis dans la CEI 62541-11.

Tableau 60 – Codes de résultats de niveau opération de HistoryUpdate

Id symbolique	Description
Bad_NotWritable	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_HistoryOperationInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_HistoryOperationUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.

5.11 Jeu de service d'appels de fonction (Method Service Set)

5.11.1 Vue générale

Les *Méthodes* représentent les appels de fonction des *Objects*. Elles sont définies dans la CEI 62541-3. Les *Méthodes* sont invoquées et retournent un résultat (couronné de succès ou

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

non) seulement à la fin. Les temps d'exécution pour les méthodes peuvent varier, selon la fonction qu'elles accomplissent.

Le *Method Service Set* (Services Méthode) définit les moyens pour invoquer des méthodes. Une *méthode* doit être un *composant* d'un *Object*. La découverte est fournie par le biais des *Services Browse* (Navigation) et *Query* (Interrogation). Les *Clients* découvrent les *méthodes* prises en charge par un *Serveur* en naviguant à la recherche des *Références* d'*Objects* propriétaires qui identifient leurs *méthodes* prises en charge.

Parce que des *Méthodes* peuvent commander un certain aspect des opérations d'une installation, l'invocation de méthodes peut dépendre des conditions d'environnement ou autres. Cela peut être particulièrement vrai lors d'une tentative d'invocation d'une méthode immédiatement que celle-ci a parachevé son exécution. Les conditions indispensables pour invoquer la *Méthode* pourraient ne pas être encore retournées à l'état qui permet à la *Méthode* de redémarrer.

5.11.2 Appel (Call)

5.11.2.1 Description

Ce *Service* est utilisé pour appeler (invoquer) une liste de *Méthodes*. Chaque appel de *Méthode* est invoqué dans le contexte d'une *Session* existante. S'il est mis fin à cette *Session*, les résultats de l'exécution de la *Méthode* ne peuvent pas être retournés au *Client* et sont éliminés. Cela est indépendant de la tâche réellement exécutée au niveau du *Serveur*.

Ce *Service* permet le passage d'arguments d'entrée et de sortie en direction/provenance de la méthode. Ces arguments sont définis par les *Properties* (Propriétés) de la *Méthode*.

5.11.2.2 Paramètres

Le Tableau 61 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 61 – Paramètres du service Call

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
methodsToCall []	CallMetthodRequest	Liste des *Méthodes* à appeler.
objectId	NodeId	Le *NodeId* doit être celui de l'*Object* ou *ObjectType* qui est la source d'une *Référence HasComponent* (ou sous-type de *Référence HasComponent*) à cette *Méthode*. Voir la CEI 62541-3 pour une description des *Objects* et de leurs *Méthodes*.
methodId	NodeId	*NodeId* de la *Méthode* à invoquer.
inputArguments []	BaseDataType	Liste de valeurs des arguments d'entrée. Une liste vide indique qu'il n'y a pas d'arguments d'entrée. La taille et l'ordre de cette liste correspondent à la taille et à l'ordre des arguments d'entrée définis par la *Property InputArguments* de la *Méthode*. Le nom, une description et le type de données de chaque argument sont définis par la structure *Argument* dans chaque élément de la *Property InputArguments* de la méthode.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
result []	CallMethodResult	Résultat pour les appels de la *Méthode*.
statusCode	StatusCode	*StatusCode* de la *Méthode* exécutée dans le serveur. Ce *StatusCode* est mis au *StatusCode* Bad_InvalidArgument si au moins un argument d'entrée a enfreint une contrainte (par exemple type de donnée erroné, valeur hors plage). Ce *StatusCode* est mis à bad *StatusCode* si l'exécution de la *Méthode* a échoué dans le *Serveur*, par exemple sur la base d'exception ou d'un HRESULT.
inputArgumentResults []	StatusCode	Liste de *StatusCodes* pour chaque *inputArgument*.
inputArgumentDiagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour chaque *inputArgument*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.
outputArguments []	BaseDataType	Liste de valeurs des arguments de sortie. Une liste vide indique qu'il n'y a pas d'arguments de sortie. La taille et l'ordre de cette liste correspondent à la taille et à l'ordre des arguments de sortie définis par la *Property OutputArguments* de la *Méthode*. Le nom, une description et le type de données de chaque argument sont définis par la structure *Argument* dans chaque élément de la *Méthode Property InputArguments*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour le *StatusCode* de *callResult*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.11.2.3 Résultats des services

Le Tableau 62 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 62 – Codes de résultats des services Call

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.11.2.4 StatusCodes

Le Tableau 60 définit les valeurs pour les paramètres *statusCode* et *operationResult* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 63 – Codes de résultats de niveau opération de Call

Id symbolique	Description
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Sert à indiquer que l'objet spécifié n'est pas valide.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat. Sert à indiquer que l'objet spécifié n'est pas valide.
Bad_ArgumentsMissing	Le client n'a pas spécifié la totalité des arguments d'entrée pour la méthode.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MethodInvalid	L'identificateur de la méthode ne se réfère pas à une méthode pour l'objet spécifié.
Bad_OutOfRange	Voir le Tableau 166 pour la description de ce code de résultat. Sert à indiquer qu'un argument d'entrée se trouve à l'extérieur de la plage acceptable.
Bad_TypeMismatch	Voir le Tableau 166 pour la description de ce code de résultat. Sert à indiquer qu'un argument d'entrée n'a pas le type de données correct.

5.12 Jeu de services des éléments surveillés (MonitoredItem Service Set)

5.12.1 Modèle pour les éléments surveillés (MonitoredItem Model)

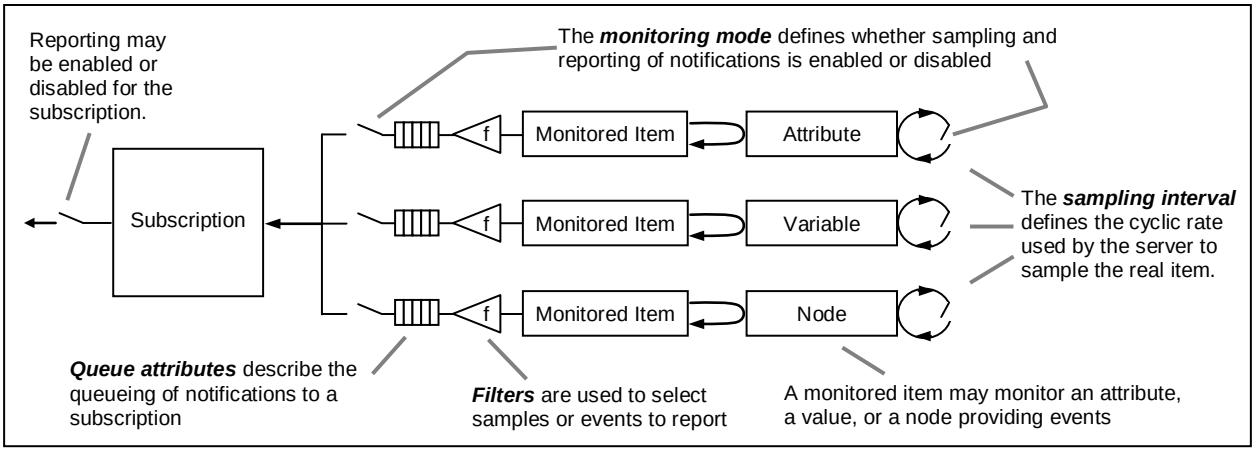
5.12.1.1 Vue générale

Les *Clients* définissent des *MonitoredItems* pour s'abonner à des données et à des *Evénements*. Chaque *MonitoredItem* identifie l'élément à surveiller et la *Souscription* à utiliser pour envoyer des *Notifications*. L'élément à surveiller peut être n'importe quel *Node Attribute*.

Les *Notifications* sont des structures de données qui décrivent l'apparition des modifications de données et d'*Evénements*. Ils sont empaquetés dans des *NotificationMessages* pour le transfert au *Client*. La *Souscription* envoie périodiquement des *NotificationMessages* selon un intervalle d'édition spécifié par l'utilisateur, et le cycle durant lequel ces messages sont envoyés est appelé cycle d'édition.

Il a été défini quatre paramètres primaires pour les *MonitoredItems* qui indiquent au *Serveur* comment l'élément doit être échantillonné, évalué et rapporté. Ces paramètres ont l'intervalle d'échantillonnage, le mode de surveillance, le filtre et le paramètre de mise en file d'attente. La Figure 15 illustre ces concepts.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



Légende

Anglais	Français
Reporting may be enabled or disabled for the subscription.	La production de rapports peut être activée ou désactivée pour l'abonnement.
Monitored Item	Elément surveillé
Attribute	Attribut
Variable	Variable
Node	Nœud
The *sampling interval* defines the cyclic rate used by the server to sample the real item.	L'*intervalle d'échantillonnage* définit la fréquence cyclique utilisée par le serveur pour échantillonner l'élément réel.
Subscription	Abonnement
The *monitoring mode* defines whether sampling and reporting of notifications are enabled or disabled	Le *mode de surveillance* définit si l'échantillonnage et les rapports de notifications sont activés ou désactivés
Filters are used to select samples or events to report	Les *filtres* sont utilisés pour sélectionner des échantillons ou des événements à consigner dans des rapports
A monitored item may monitor an attribute, a value, or a node providing events	Un élément surveillé peut surveiller un attribut, une valeur ou un nœud fournissant des événements
Queue attributes describe the queueing of notifications to a subscription	Les *attributs de file d'attente* décrivent la mise en file d'attente de notifications destinées à un abonnement

Figure 15 – Modèle MonitoredItem

Les *Attributs* autres que l'*Attribute Value* sont seulement surveillés pour détecter un changement de valeur. Le filtre n'est pas utilisé pour ces *Attributs*. Toute modification de valeur pour ces *Attributs* entraîne la création d'une *Notification*.

L'*Attribute Value* (Valeur Attribut) est utilisé pour surveiller des *Variables*. Les valeurs des *Variables* sont surveillées pour détecter une modification de valeur ou une modification de leur statut. Les filtres définis dans la présente norme (voir 7.16.2) et dans la CEI 62541-8 sont utilisés pour déterminer si la modification de la valeur est suffisamment forte pour entraîner la création d'une *Notification* pour cette *Variable*.

Les *Objects* et les vues peuvent être utilisés pour surveiller les *Evénements*. Les *Evénements* sont disponibles uniquement à partir *des Noeuds* pour lesquels le bit *SubscribeToEvents* de

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

l'*Attribute EventNotifier* est positionné. Le filtre défini dans la présente norme (voir 7.16.3) est utilisé pour déterminer si un *Evénement* reçu *du Noeud* est envoyé au *Client*. Le filtre permet aussi de sélectionner des champs de l'*EventType* qui seront contenus dans l'*Evénement* tels que *EventId*, *EventType*, *SourceNode*, *Time* et *Description*.

La CEI 62541-3 décrit le modèle *Evénement* et les *EventTypes* de base.

Les *Properties* des *EventTypes* de base et la représentation des *EventTypes* de base dans *l'Espace d'Adresses* sont spécifiées dans la CEI 62541-5.

5.12.1.2 Intervalle d'échantillonnage

Chaque *MonitoredItem* créé par le *Client* reçoit un intervalle d'échantillonnage qui lui est affecté et qui est soit hérité de l'intervalle d'édition de la *Souscription*, soit défini spécifiquement pour se substituer à cette fréquence. L'intervalle d'échantillonnage indique la fréquence la plus rapide à laquelle il convient que le *Serveur* échantillonne sa source sous-jacente pour les modifications de données.

Un *Client* doit définir un intervalle d'échantillonnage de 0 s'il s'abonne à des *Evénements*.

L'intervalle d'échantillonnage affecté définit une fréquence cyclique de «meilleur effort» que le *Serveur* utilise pour échantillonner l'élément à partir de sa source. «Meilleur effort» signifie dans ce contexte que le *Serveur* fait de son mieux pour échantillonner à cette fréquence. L'échantillonnage à des fréquences plus rapides que celle-ci est acceptable, mais n'est pas indispensable pour répondre aux besoins du *Client*. La façon dont le *Serveur* traite de la fréquence d'échantillonnage et la fréquence à laquelle il sonde effectivement ses sources de données en interne sont un détail d'implémentation du *Serveur*. Cependant, le temps entre les valeurs retournées au *Client* doit être supérieur ou égal à l'intervalle d'échantillonnage.

Le *Client* peut aussi spécifier 0 pour l'intervalle d'échantillonnage, qui indique qu'il convient pour le *Serveur* d'utiliser la fréquence pratique la plus rapide. Il est escompté que les *Serveurs* prendront en charge uniquement un jeu limité d'intervalles d'échantillonnage pour optimiser leur fonctionnement. Si l'intervalle exact demandé par le *Client* n'est pas pris en charge par le *Serveur*, le *Serveur* affecte au *MonitoredItem* l'intervalle le plus approprié tel que déterminé par le *Serveur*. Il retourne au *Client* cet intervalle affecté. L'*Object Server* Capabilities (l'objet Fonctions du serveur) défini dans la CEI 62541-5 identifie les intervalles d'échantillonnage pris en charge par le *Serveur*.

Le *Serveur* peut prendre en charge les données qui sont recueillies sur la base d'un modèle d'échantillonnage ou générées sur la base d'un modèle piloté par des exceptions. L'intervalle d'échantillonnage pris en charge le plus rapide peut être égal à 0, ceci indique que l'élément de donnée est piloté par des exceptions au lieu d'être échantillonné périodiquement. Piloté par des exceptions signifie que le système sous-jacent ne nécessite pas d'échantillonnage et rapporte les changements apportés aux données.

Le *Client* peut utiliser des valeurs révisées de l'intervalle d'échantillonnage comme suggestion pour l'établissement de l'intervalle d'édition ainsi que le compte d'entretien d'une *Souscription*. Si, par exemple, le plus petit intervalle d'échantillonnage révisé pour les *MonitoredItems* est 5 s, il convient que le temps avant qu'un entretien ne soit envoyé soit supérieur à 5 s.

A noter que, dans de nombreux cas, le *Serveur* OPC UA fournit l'accès à un système découplé et n'a donc aucune connaissance de la logique de mise à jour de données. Dans ce cas, même si le *Serveur* OPC UA échantillonne à la fréquence négociée, les données pourraient être mises à jour par le système sous-jacent à une fréquence beaucoup plus lente. Dans ce cas, les modifications ne peuvent être détectées qu'à cette fréquence plus lente.

Si le comportement par lequel le système sous-jacent met à jour l'élément est connu, il sera disponible via l'*Attribut MinimumSamplingInterval* défini dans la CEI 62541-3. Si le *Serveur*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

spécifie une valeur pour l'*Attribut MinimumSamplingInterval*, il doit toujours retourner un *revisedSamplingInterval* qui est supérieur ou égal au *MinimumSamplingInterval* si le *Client* s'abonne à l'*Attribute Value*.

Il convient que les *Clients* sachent que l'échantillonnage par le *Serveur* OPC UA et le cycle de mise à jour du système sous-jacent ne sont pas, en général, synchronisés. Cela peut introduire des retards supplémentaires dans la détection des modifications, comme illustré à la Figure 16.

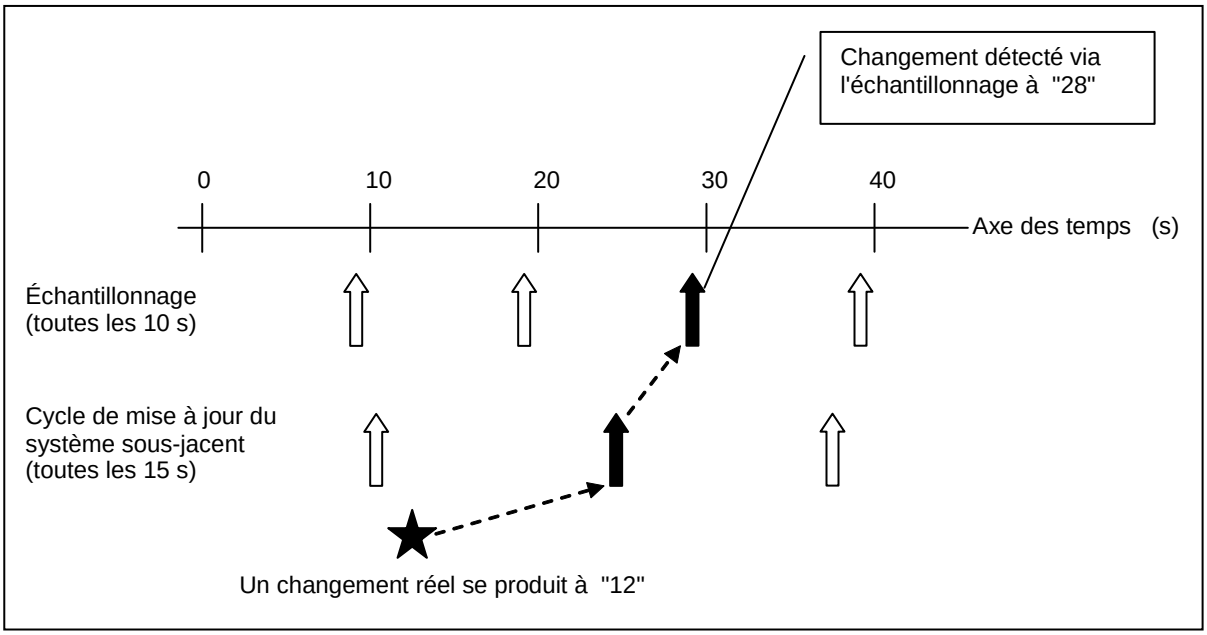


Figure 16 – Retard type pour détecter des modifications

5.12.1.3 Mode de surveillance

Le paramètre mode de surveillance est utilisé pour activer et désactiver l'échantillonnage d'un *MonitoredItem* et aussi pour permettre d'activer et de désactiver de façon indépendante la production de rapports de *Notifications*. Cette fonction permet de configurer un *MonitoredItem* pour échantillonner, échantillonner et rapporter ou ni l'un ni l'autre. La désactivation de l'échantillonnage ne change les valeurs d'aucuns des autres paramètres de *MonitoredItem*, tels que son intervalle d'échantillonnage.

Lorsqu'un *MonitoredItem* est activé (c'est-à-dire lorsque le *MonitoringMode* est passé de *DISABLED* à *SAMPLING* ou à *REPORTING*) ou qu'il est créé dans l'état activé, le *Serveur* doit signaler le premier échantillon dès que possible dans le rapport et l'heure de cet échantillon devient le point de départ pour le prochain intervalle d'échantillonnage.

5.12.1.4 Filtre

Chaque fois qu'un *MonitoredItem* est échantillonné, le *Serveur* évalue l'échantillon en utilisant le filtre défini pour le *MonitoredItem*. Le paramètre filter défini les critères que le *Serveur* utilise pour déterminer s'il faut générer une *Notification* pour l'échantillon. Le type de filtre dépend du type d'élément qui est surveillé. Par exemple, le *DataChangeFilter* et l'*AggregateFilter* sont utilisés pour surveiller les *Variable Values* (Valeurs de variables) et l'*EventFilter* est utilisé pour surveiller des *Evénements*. L'échantillonnage et l'évaluation, y compris l'utilisation de filtres, sont décrits dans la présente norme. Des filtres supplémentaires peuvent être définis dans d'autres parties de la présente série de normes.

5.12.1.5 Paramètres de mise en file d'attente

Si l'échantillon satisfait aux critères de filtrage, une *Notification* est générée et mise en file d'attente pour le transfert par la *Souscription*. La taille de la file d'attente est définie lorsque

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

MonitoredItem est créé. Lorsque la file d'attente est pleine et qu'une nouvelle *Notification* est reçue, le *Serveur* soit élimine la plus ancienne *Notification* et place la nouvelle dans la file d'attente, soit rejette simplement la nouvelle. A sa création, le *MonitoredItem* est configuré pour l'une de ces politiques de rejet. Si une Notification est rejetée pour une *DataValue*, le bit *Overflow* (dépassement de capacité) dans la partie *InfoBits* du *statusCode* de la *DataValue* est positionné.

Si la taille de la file d'attente est 1 et la politique de rejet est d'éliminer la notification la plus ancienne, la file d'attente devient un tampon qui contient en permanence la *Notification* la plus récente. Dans ce cas, si l'intervalle d'échantillonnage du *MonitoredItem* est plus court que l'intervalle d'édition d'une *Souscription*, le *MonitoredItem* sera en suréchantillonnage et le *Client* recevra toujours la valeur la plus récente.

D'autre part, le *Client* peut souhaiter s'abonner à un flux continu de *Notifications* sans le moindre trou, mais ne souhaitent pas qu'elles soient rapportées à l'intervalle d'échantillonnage. Dans ce cas, le *MonitoredItem* met en place une file d'attente suffisamment longue pour contenir toutes les *Notifications* générées entre deux cycles d'édition consécutifs. Puis, à chaque cycle d'édition, la *Souscription* envoie au *Client* toutes les *Notifications* mises en file d'attente pour le *MonitoredItem*. Le *Serveur* doit retourner les *Notifications* pour tout élément particulier dans l'ordre dans lequel elles sont stokées dans la file d'attente.

Le *Serveur* peut échantillonner à une fréquence plus rapide que l'intervalle d'échantillonnage pour prendre en charge d'autres *Clients*; il convient que le *Client* attende seulement les valeurs à l'intervalle d'échantillonnage négocié. Le *Serveur* peut délivrer moins de valeurs que ne le dicte l'intervalle d'échantillonnage, sur la base des contraintes du filtre et de l'implémentation. Si un *DataChangeFilter* est configuré pour un *MonitoredItem*, il est toujours appliqué à la valeur la plus récente dans la file d'attente en comparaison à l'échantillon courant.

Si, par exemple, l'*AbsoluteDeadband* dans le *DataChangeFilter* est «10», la file d'attente pourrait être constituée de valeur dans l'ordre suivant:

- 100
- 111
- 101
- 89
- 100

La mise des données en file d'attente peut conduire à un comportement inattendu lorsqu'un filtre *Deadband* est utilisé et le nombre de modifications rencontrées est supérieur au nombre de valeurs qui peuvent être maintenues. Il existe une possibilité réaliste qu'en raison de la politique de rejet «`discardOldest=TRUE`», la première nouvelle valeur dans la file d'attente puisse ne pas dépasser la limite de *Deadband* de la valeur précédente envoyée au *Client*.

La taille de la file d'attente est la valeur maximale prise en charge par le *Serveur* lors de la surveillance d'*Evénements*. Dans ce cas, le *Serveur* a la responsabilité du tampon d'*Evénements*. Si des *Evénements* sont perdus, un *Evénement* du type *EventQueueOverflowEventQueueOverflowEventType* est généré et placé. Cet Evénement est généré lorsque le premier Evénement doit être éliminé sur un MonitoredItem s'abonnant à des Evénements. Il est mis dans la File d'attente du MonitoredItem en plus de la taille de la File d'attente définie pour ce MonitoredItem sans rejeter aucun autre Evénement. Si discardOldest est mis à TRUE, il est placé au début de la file d'attente et n'est jamais rejeté, autrement à la fin. Un *Serveur* combiné ne doit pas transmettre un tel *Evénement*. Il doit être géré comme les autres scénarios d'erreur de connexion.

5.12.1.6 Modèle de déclenchement

Le *Service MonitoredItems* permet d'ajouter des éléments qui sont signalés dans un rapport seulement lorsqu'un autre élément est déclenché (l'élément de déclanchement). Cela est réalisé en créant des liens entre les éléments déclenchés et les éléments à rapporter. Le mode de surveillance des éléments à rapporter est mis en échantillonnage seul et, de ce fait,

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

il échantillonnera et mettra les *Notifications* en file d'attente sans les signaler dans un rapport. La Figure 17 illustre ce concept.

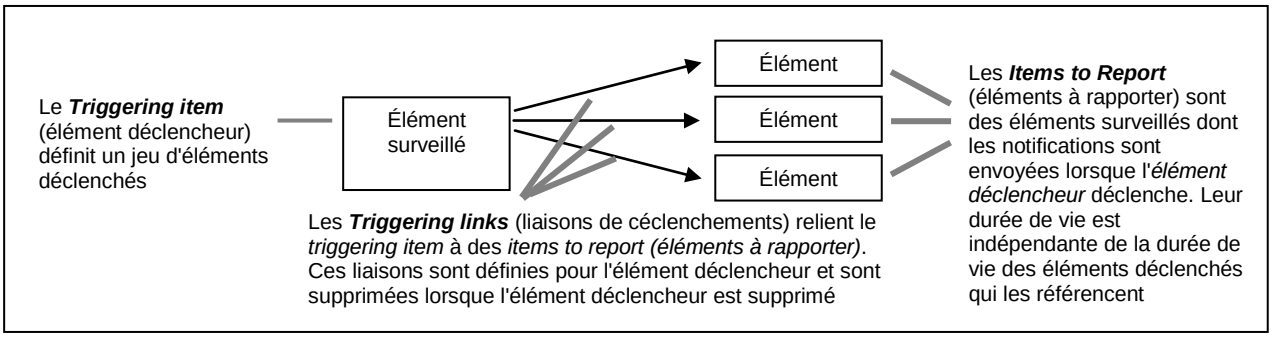


Figure 17 – Modèle de déclenchement

Le mécanisme de déclenchement est une caractéristique utile qui permet aux *Clients* de réduire le volume de données en configurant certains éléments pour qu'ils échantillonnent fréquemment mais se limitent seulement à produire un rapport lorsqu'un certain autre *Evénement* se produisent.

Les comportements de déclenchement suivants sont spécifiés:

a) Le mode monitoring (de surveillance) de l'élément déclencheur indique que la production de rapport (reporting) est désactivée. Dans ce cas, l'élément déclencheur n'est pas signalé dans un rapport lorsque l'élément déclencheur déclenche.

b) Le mode monitoring (de surveillance) de l'élément déclencheur indique que la production de rapport (reporting) est activée. Dans ce cas, l'élément déclencheur est signalé dans un rapport lorsque l'élément déclencheur déclenche.

c) Le mode monitoring (de surveillance) de l'élément à rapporter indique que la production de rapport (reporting) est désactivée. Dans ce cas, l'élément à rapporter est signalé dans un rapport lorsque l'élément déclencheur déclenche.

d) Le mode monitoring (de surveillance) de l'élément à rapporter indique que la production de rapport (reporting) est activée. Dans ce cas, l'élément à rapporter est signalé dans le rapport une seule fois (lorsque l'élément à rapporter déclenche) et, de ce fait, l'élément déclencheur est effectivement ignoré.

Les *Clients* créent et suppriment des liaisons de déclenchement entre un élément déclencheur et un ensemble d'éléments à rapporter. Si le *MonitoredItem* qui représente un élément à rapporter est supprimé avant que sa liaison associée ne soit supprimée, la liaison de déclenchement est également supprimée, mais l'élément déclencheur n'est pas altéré.

Il convient de ne pas confondre la suppression d'un *MonitoredItem* avec le retrait de l'*Attribut* qu'il surveille. Si le Noeud qui contient l'*Attribut* surveillé est supprimé, le *MonitoredItem* génère une *Notification* avec un *StatusCode Bad_UnknownNodeId* qui indique la suppression, mais le *MonitoredItem* n'est pas supprimé.

5.12.2 Créer des éléments de controle (CreateMonitoredItems)

5.12.2.1 Description

Ce *Service* est utilisé pour créer et ajouter un ou plusieurs *MonitoredItems* à une *Souscription*. Un *MonitoredItem* est supprimé automatiquement par le *Serveur* lorsque la *Souscription* est supprimée. La suppression d'un *MonitoredItem* entraîne celle de l'ensemble complet des liaisons d'éléments déclenchés, mais n'a pas d'effet sur les *MonitoredItems* référencés par les éléments déclenchés.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'appel à répétions du *Service CreateMonitoredItems* pour ajouter chaque fois un petit nombre de *MonitoredItems* peut affecter défavorablement la performance du *Serveur*. Il est préferable que les *Clients* ajoutent un jeu complet de *MonitoredItems* a la *Souscription* chaque fois que possible.

Lorsqu'un *MonitoredItem* est ajouté, le *Serveur* exécute un traitement d'initialisation pour celui-ci. Le traitement d'initialisation est défini par le type de *Notification* de l'élément surveillé. Les types de *Notification* sont spécifiés dans la présente norme et dans les parties qui spécifient les types d'accès, telles que la CEI 62541-8. Voir l'Article 4 de la CEI/TR 62541-1 pour la description des Parties Types d'accès

Lorsqu'un utilisateur ajoute un élément surveillé dont l'accès en lecture est refusé à l'utilisateur, l'opération d'ajout de l'élément doit réussir et le mauvais statut Bad_NotReadable ou Bad_UserAccessDenied doit être retourné dans la réponse Publish (Editer). Il s'agit du même comportement que dans le cas où les droits d'accès sont modifiés après l'appel à *CreateMonitoredItem*. Si les droits d'accès passent aux droits de lecture, le *Serveur* doit commencer à envoyer des données pour le *MonitoredItem*.

Le retour des informations de réglage des diagnostic dans l'en-tête de demande des *CreateMonitoredItems* ou le dernier *Service ModifyMonitoredItems* est appliqué aux *Monitored Items* (aux éléments surveillés) et il sert de réglages d'informations de diagnostic lors de l'envoi de la Notifications dans la réponse *Publish*.

5.12.2.2 Paramètres

Le Tableau 64 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 64 – Paramètres du service CreateMonitoredItems

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* qui signalera dans un rapport les *Notifications* pour ce *MonitoredItem* (voir 7.13 pour la définition d'*IntegerId*).
timestampsToReturn	enum TimestampsToReturn	Une énumération qui spécifie les *Attributs* d'horodatage à transmettre pour chaque *MonitoredItem*. L'énumération *TimestampsToReturn* est définie en 7.34. Lors de la surveillance d'*Evénements*, cela s'applique seulement aux champs *Evénement* qui sont du type *DataValue*.
itemsToCreate []	MonitoredItemCreateRequest	Une liste des *MonitoredItems* à créer et à affecter a la *Souscription* spécifié.
itemToMonitor	ReadValueId	Identifie un élément dans l'Espace d'adresses à surveiller. Pour surveiller afin de détecter des *Evénements*, l'élément *attributeId* de la structure *ReadValueId* est l'identificateur de l'*Attribute EventNotifier*. Le type *ReadValueId* est défini en 7.23.
monitoringMode	enum MonitoringMode	Le mode de surveillance à établir pour le *MonitoredItem*. L'énumération *MonitoringMode* est définie en 7.17.
requestedParameters	MonitoringParameters	Les paramètres de surveillance demandés. Les *Serveurs* négocient les valeurs de ces paramètres sur la base d'une *Souscription* et des fonctions du *Serveur*. Le type *MonitoringParameters* est défini en 7.15.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	MonitoredItemCreateResult	Liste de résultats pour les *MonitoredItems* à créer. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *itemsToCreate*.
statusCode	StatusCode	*StatusCode* pour le *MonitoredItem* à créer (voir 7.33 pour la définition de *StatusCode*).
monitoredItemId	IntegerId	Identificateur affecté par le *Serveur* pour le *MonitoredItem* (voir 7.13 pour la définition d'*IntegerId*). Cet identificateur est unique dans la *Souscription*, mais pourrait ne pas être unique dans le *Serveur* ou dans une *Session*. Ce paramètre est présent seulement si le *statusCode* indique que le *MonitoredItem* a été créé avec succès.
revisedSamplingInterval	Duration	L'intervalle d'échantillonnage réel que le *Serveur* utilisera. Cette valeur est fondée sur plusieurs facteurs, y compris les fonctions du système sous-jacent. Le *Serveur* doit toujours retourner un *revisedSamplingInterval* qui est supérieur ou égal au *requestedSamplingInterval*. Si le *requestedSamplingInterval* est supérieur à l'intervalle d'échantillonnage maximal pris en charge par le *Serveur*, c'est l'intervalle d'échantillonnage maximal qui est retourné.
revisedQueueSize	Counter	La taille réelle de file d'attente que le *Serveur* utilisera.
filterResult	Extensible Parameter MonitoringFilterResult	Contient toutes les éventuelles valeurs de paramètre révisées ou tous les éventuels résultats d'erreur associés au *MonitoringFilter* spécifié dans la demande. Ce paramètre peut être omis s'il ne s'est pas produit d'erreurs. Le type de paramètre *MonitoringFilterResult* est un type de paramètre extensible spécifié en 7.16.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *MonitoredItems* à créer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *itemsToCreate*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.12.2.3 Résultats des services

Le Tableau 65 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 65 – Codes de résultats des services CreateMonitoredItems

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TimestampsToReturnInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.12.2.4 StatusCodes

Le Tableau 66 définit les valeurs de niveau opération pour le paramètre *statusCode* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 66 – Codes de résultats de niveau opération de CreateMonitoredItems

Id symbolique	Description
Bad_MonitoringModeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_AttributeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_IndexRangeNoData	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_DataEncodingUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MonitoredItemFilterInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_MonitoredItemFilterUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_FilterNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.

5.12.3 Modifier des éléments de controle (ModifyMonitoredItems)

5.12.3.1 Description

Ce *Service* est utilisé pour modifier les *MonitoredItems* d'une *Souscription*. Les modifications apportées à l'intervalle d'échantillonnage et au filtre prennent effet au commencement du prochain intervalle d'échantillonnage (la prochaine fois que le temporisateur d'échantillonnage expire).

Le retour des informations de réglage des diagnostics dans l'en-tête de demande de *CreateMonitoredItems* ou le dernier *Service ModifyMonitoredItems* est appliqué aux *Monitored Items* (aux éléments surveillés) et sert de réglages d'informations de diagnostic lors de l'envoi de Notifications dans la réponse *Publish*.

5.12.3.2 Paramètres

Le Tableau 67 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 67 – Paramètres du service ModifyMonitoredItems

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* utilisé pour qualifier le *monitoredItemId* (voir 7.13 pour la définition de *IntegerId*).
timestampsToReturn	enum TimestampsToReturn	Une énumération qui spécifie les attributs d'horodatage à transmettre pour chaque *MonitoredItem* à modifier. L'énumération *TimestampsToReturn* est définie en 7.34. Lors de la surveillance d'*Evénements*, cela s'applique seulement aux champs *Evénement* qui sont du type *DataValue*.
itemsToModify []	MonitoredItemModifyRequest	La liste des *MonitoredItems* à modifier.
monitoredItemId	IntegerId	Identificateur affecté par le *Serveur* pour le *MonitoredItem*.
requestedParameters	MonitoringParameters	Les valeurs demandées pour les paramètres de surveillance. Le type *MonitoringParameters* est défini en 7.14.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	MonitoredItemModifyResult	Liste de résultats pour les *MonitoredItems* à modifier. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *itemsToModify*.
statusCode	StatusCode	*StatusCode* pour le *MonitoredItem* à modifier (voir 7.33 pour la définition de *StatusCode*).
revisedSamplingInterval	Duration	L'intervalle d'échantillonnage réel que le *Serveur* utilisera. Le *Serveur* retourne la valeur qu'il utilisera réellement pour l'intervalle d'échantillonnage. Cette valeur est fondée sur plusieurs facteurs, y compris les fonctions du système sous-jacent. Le *Serveur* doit toujours retourner un *revisedSamplingInterval* qui est supérieur ou égal au *requestedSamplingInterval*. Si le *requestedSamplingInterval* est supérieur à l'intervalle d'échantillonnage maximal pris en charge par le *Serveur*, c'est l'intervalle d'échantillonnage maximal qui est retourné.
revisedQueueSize	Counter	La taille réelle de file d'attente que le *Serveur* utilisera.
filterResult	Extensible Parameter MonitoringFilterResult	Contient toutes les éventuelles valeurs de paramètre révisées ou tous les éventuels résultats d'erreurs associés au *MonitoringFilter* spécifié dans la demande. Ce paramètre peut être omis s'il ne s'est pas produit d'erreurs. Le type de paramètre *MonitoringFilterResult* est un type de paramètre extensible spécifié en 7.16.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *MonitoredItems* à modifier (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *itemsToModify*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.12.3.3 Résultats des services

Le Tableau 68 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 68 – Codes de résultats des services ModifyMonitoredItems

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TimestampsToReturnInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.12.3.4 StatusCodes

Le Tableau 69 définit les valeurs pour le paramètre *statusCode* de niveau opération qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 69 – Codes de résultats de niveau opération de ModifyMonitoredItems

Id symbolique	Description
Bad_MonitoredItemIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_MonitoredItemFilterInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_MonitoredItemFilterUnsupported	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_FilterNotAllowed	Voir le Tableau 165 pour la description de ce code de résultat.

5.12.4 Gérer des modes de controle (SetMonitoringMode)

5.12.4.1 Description

Ce *Service* est utilisé pour établir le mode de surveillance pour un ou plusieurs *MonitoredItems* d'une *Souscription*. Le fait de mettre le mode à DISABLED entraîne la suppression de toutes les *Notifications* placées en file d'attente.

5.12.4.2 Paramètres

Le Tableau 70 définit les paramètres du *Service*.

Tableau 70 – Paramètres du service SetMonitoringMode

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* utilisé pour qualifier les *monitoredItemIds* (voir 7.13 pour la définition d'*IntegerId*).
monitoringMode	enum MonitoringMode	Le mode de surveillance à établir pour les *MonitoredItems*. L'énumération *MonitoringMode* est définie en 7.17.
monitoredItemIds []	IntegerId	Liste des identificateurs affectés par le *Serveur* pour les *MonitoredItems*.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *MonitoredItems* à activer/désactiver (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *monitoredItemIds*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *MonitoredItems* à activer/désactiver (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *monitoredItemIds*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.12.4.3 Résultats des services

Le Tableau 71 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 71 – Codes de résultats des services SetMonitoringMode

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MonitoringModeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.12.4.4 StatusCodes

Le Tableau 72 définit les valeurs de niveau opération pour le paramètre *statusCode* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 72 – Codes de résultats de niveau opération de SetMonitoringMode

Valeur	Description
Bad_MonitoredItemIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.12.5 Gérer des déclenchements (SetTriggering)

5.12.5.1 Description

Ce *Service* est utilisé pour créer et supprimer des liaisons de déclenchement pour un élément déclencheur. L'élément déclencheur et les éléments à rapporter doivent appartenir à la même *Souscription*.

Chaque liaison de déclenchement relie un élément déclencheur à un élément à rapporter. Chaque liaison est représentée par l'identificateur de *MonitoredItem* pour l'élément à rapporter. Un code d'erreur est retourné si cet identificateur n'est pas valide.

5.12.5.2 Paramètres

Le Tableau 73 définit les paramètres du *Service*.

Tableau 73 – Paramètres du service SetTriggering

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* qui contient l'élément déclencheur et les éléments à rapporter (voir 7.13 pour la définition d'*IntegerId*).
triggeringItemId	IntegerId	Identificateur affecté par le *Serveur* pour le *MonitoredItem* utilisé comme l'élément déclencheur.
linksToAdd []	IntegerId	La liste des identificateurs affectés par le *Serveur* des éléments à rapporter qui doivent être ajoutés comme liaisons de déclenchement.
linksToRemove []	IntegerId	La liste des identificateurs affectés par le *Serveur* des éléments à rapporter pour les liaisons de déclenchement à supprimer.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
addResults []	StatusCode	Liste des *StatusCodes* pour les éléments à ajouter (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre *linksToAdd* spécifié dans la demande.
addDiagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les liaisons à ajouter (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *linksToAdd*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.
removeResults []	StatusCode	Liste des *StatusCodes* pour les éléments à supprimer. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre *linksToDelete* spécifié dans la demande.
removeDiagnosticInfos []	Diagnostic Info	Liste d'informations de diagnostic pour les liaisons à supprimer. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *linksToDelete*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.12.5.3 Résultats des services

Le Tableau 74 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au 7.33.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 74 – Codes de résultats des services SetTriggering

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MonitoredItemIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.12.5.4 StatusCodes

Le Tableau 75 définit les valeurs pour les paramètres results qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 75 – Codes de résultats de niveau opération de SetTriggering

Id symbolique	Description
Bad_MonitoredItemIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.12.6 Supprimer des items (DeleteMonitoredItems)

5.12.6.1 Description

Ce *Service* est utilisé pour retirer un ou plusieurs *MonitoredItems* d'une *Souscription*. Lorsqu'un *MonitoredItem* est supprimé, ses liaisons d'éléments déclenchés sont supprimées aussi.

Le retrait réussi d'un *MonitoredItem* pourrait toutefois ne pas retirer les *Notifications* pour le *MonitoredItem* qui sont en cours d'envoi par la *Souscription*. Par conséquent, les *Clients* peuvent recevoir des *Notifications* pour le *MonitoredItem* après qu'ils aient reçu une réponse positive indiquant que le *MonitoredItem* a été supprimé.

5.12.6.2 Paramètres

Le Tableau 76 définit les paramètres du *Service*.

Tableau 76 – Paramètres du service DeleteMonitoredItems

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* qui contient les *MonitoredItems* à supprimer (voir 7.13 pour la définition de *IntegerId*).
monitoredItemIds []	IntegerId	Liste des identificateurs affectés par le *Serveur* pour les *MonitoredItems* à supprimer.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *MonitoredItems* à supprimer (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *monitoredItemIds*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *MonitoredItems* à supprimer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *monitoredItemIds*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.12.6.3 Résultats des services

Le Tableau 77 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 77 – Codes de résultats des services DeleteMonitoredItems

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.12.6.4 StatusCodes

Le Tableau 78 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 78 – Codes de résultats de niveau opération de DeleteMonitoredItems

Id symbolique	Description
Bad_MonitoredItemIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.

5.13 Subscription Service Set

5.13.1 Abonnement (SubscriptionModel)

5.13.1.1 Description

Les *Souscriptions* sont utilisées pour rapporter des *Notifications* au *Client*. Leur comportement général est récapitulé ci-après. Leur comportement précis est décrit en 5.13.1.2.

a) Les *Souscriptions* ont des *MonitoredItems* qui leur sont affectés par le *Client*. Les *MonitoredItems* génèrent des *Notifications* qui doivent faire l'objet d'un rapport au *Client* par la *Souscription* (voir 5.12.1 pour une description des *MonitoredItems*).

b) Les *Souscriptions* ont un intervalle d'édition. L'intervalle d'édition d'une *Souscription* définit la fréquence cyclique à laquelle la *Souscription* s'exécute. Chaque fois qu'elle s'exécute, il tente d'envoyer un *NotificationMessage* au *Client*. Les *NotificationMessages* contiennent des *Notifications* qui n'ont pas encore été rapportées au *Client*.

c) Les *NotificationMessages* sont envoyés au *Client* en réponse à des demandes *Publish*. Les demandes *Publish* sont normalement placées en file d'attente dans la *Session* à mesure qu'elles sont reçues, et une est sortie de la file d'attente et traitée par un abonnement relatif à cette *Session* à chaque cycle d'édition, s'il y a des *Notifications* à rapporter. Lorsqu'il n'y en a pas, la demande *Publish* est sortie de la file d'attente de la *Session* et le *Serveur* attend le prochain cycle et vérifie de nouveau la présence de *Notifications*.

d) Au début d'un cycle, s'il y a des *Notifications* à envoyer mais s'il n'y a pas de demandes *Publish* placées en file d'attente, le *Serveur* entre dans l'état d'attente (wait) pour qu'une demande *Publish* soit reçue. Lorsqu'une demande est reçue, elle est traitée immédiatement sans attendre le prochain cycle d'édition.

e) Les *NotificationMessages* sont identifiés de manière unique par des numéros de séquence qui permettent à des *Clients* de détecter des *Messages* manqués. L'intervalle d'édition définit aussi l'intervalle d'échantillonnage par défaut pour ses *MonitoredItems*.

f) Les *Souscriptions* ont un compteur d'entretien qui compte le nombre de cycles d'édition consécutifs dans lesquels il n'y a pas de *Notifications* à rapporter au *Client*. Lorsque le compte d'entretien maximal est atteint, une demande *Publish* est sortie de la file d'attente et utilisée pour retourner le *Message* d'entretien. Ce *Message* d'entretien informe le *Client* que la *Souscription* est encore active. Chaque *Message* d'entretien est une réponse à une demande Publish dans laquelle le paramètre *notificationMessage* ne contient pas de Notifications et qui contient le numéro de séquence du prochain *NotificationMessage* qui doit être envoyé. Dans les paragraphes qui suivent, le terme *NotificationMessage* se réfère à une réponse à une demande Publish dans laquelle le paramètre *notificationMessage* contient réellement une ou plusieurs *Notifications*, par opposition à un *keep-alive Message* (Message d'entretien) dans lequel ce paramètre ne contient pas de *Notifications*. Le compte d'entretien maximal est établi par le *Client* pendant la création

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

d'une *Souscription* et peut être modifié ultérieurement en utilisant le *Service ModifySubscription*. D'une manière similaire au traitement des *Notifications* décrit en c) ci-dessus, s'il n'y a pas de demandes *Publish* placées en file d'attente, le *Serveur* attend la réception de la prochaine demande et envoie immédiatement le message d'entretien sans attendre le prochain cycle d'édition.

g) L'édition par une *Souscription* peut être activée ou désactivée par le *Client* à sa création ou ultérieurement en utilisant le *Service SetPublishingMode*. La désactivation force la *Souscription* à cesser d'envoyer des *NotificationMessages* au *Client*. Cependant, la *Souscription* continue à s'exécuter de façon cyclique et continue d'envoyer des *Messages* d'entretien au *Client*.

h) Les *Souscriptions* ont un compteur de durée de vie qui compte le nombre de cycles d'édition consécutifs dans lesquels il n'y a pas de *demandes Publish* reçues du *Client*. Lorsque ce compteur atteint la valeur calculée pour la durée de vie d'une *Souscription* en fonction du paramètre MaxKeepAliveCount du *Service CreateSubscription* (5.13.2), la *Souscription* est fermée. La fermeture d'une *Souscription* entraine la suppression de ses *MonitoredItems*. De plus, le *Serveur* doit émettre un *notificationMessage* de *StatusChangeNotification* avec le code de statut Bad_Timeout. Le type *StatusChangeNotification notificationMessage* est défini en 7.19.4.

i) Les *Souscriptions* maintiennent une file d'attente de transmission des *NotificationMessages* envoyés. Les *NotificationMessages* sont retenus dans cette file d'attente jusqu'à ce qu'ils soient acquittés ou jusqu'à ce qu'ils aient été dans la file d'attente pendant au moins un intervalle d'entretien. Les *Clients* sont tenus d'acquitter les *NotificationMessages* à mesure qu'ils sont reçus.

Le numéro de séquence est un entier non signé de 32 bits qui est incrémenté de 1 pour chaque *NotificationMessage* envoyé. La valeur 0 n'est jamais utilisée comme numéro de séquence. Le premier *NotificationMessage* envoyé sur une *Souscription* a le numéro de séquence 1. Si le numéro de séquence atteint son maximum, il repasse à 1.

Lorsqu'une *Souscription* est créée, le premier *Message* est envoyé à la fin du premier cycle d'édition pour signaler au *Client* que la *Souscription* est opérationnelle. Un *NotificationMessage* est envoyé s'il y a des *Notifications* prêtes à être rapportées. S'il n'y en a pas, il est envoyé à la place un *Message* d'entretien qui contient un numéro de séquence de 1, indiquant que le premier *NotificationMessage* n'a pas encore été envoyé. C'est la seule fois qu'un *Message* d'entretien est envoyé sans attendre d'avoir atteint le compte d'entretien maximal, comme spécifié en f) ci-dessus.

La valeur du numéro de séquence n'est jamais réinitialisée au cours de la durée de vie d'une *Souscription*. Par conséquent, ce même numéro de séquence ne doit jamais être réutilisé par une *Souscription* avant que plus de quatre milliards de *NotificationMessages* aient été envoyés. A une fréquence continue de mille *NotificationMessages* par seconde sur une *Souscription* donnée, il faudrait grosso modo cinquante jours pour que le même numéro de séquence soit réutilisé. Cela permet aux *Clients* de traiter en toute sécurité les numéros de séquence comme étant uniques.

Les numéros de séquence sont également utilisés par les *Clients* pour acquitter réception des *NotificationMessages*. Les demandes *Publish* permettent au *Client* d'acquitter toutes les *Notifications* jusqu'à un numéro de séquence spécifique et permettent d'acquitter le numéro de séquence du dernier *NotificationMessage* reçu. Un ou plusieurs trous peuvent exister entre elles. Les acquittements permettent au *Serveur* d'effacer des *NotificationMessages* de sa file d'attente de retransmission.

Les *Clients* peuvent demander la retransmission de *NotificationMessages* sélectionnés en utilisant le *Service* Republish. Ce *Service* retourne le *Message* demandé.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.1.2 Table d'états

La table d'états décrit le fonctionnement d'une *Souscription*. Le modèle de fonctionnements est décrit par cette table d'états. Cette description s'applique lorsque l'édition est activée ou désactivée pour la *Souscription*.

Après la création d'une *Souscription*, le *Serveur* lance la minuterie d'édition et la redémarre chaque fois qu'elle expire. Si la minuterie expire le nombre de fois défini pour la durée de vie d'une *Souscription* sans avoir reçu de demande du *Service* de *Souscription* de la part du *Client*, la *Souscription* suppose que le *Client* n'est plus présent et se termine.

Les *Clients* envoient des demandes *Publish* aux *Serveurs* pour recevoir des *Notifications*. Les demandes *Publish* ne sont dirigées à aucune *Souscription* spécifique et, par conséquent, peuvent être utilisées par n'importe quelle *Souscription*. Chacune contient des acquittements pour une ou plusieurs *Souscriptions*. Ces acquittements sont traités lorsque la demande *Publish* est reçue. Le *Serveur* met alors la demande dans une file d'attente partagée par toutes les *Souscriptions*, sauf dans les cas suivants:

a) La réponse *Publish* précédente indiquait qu'il y avait encore d'autres *Notifications* prêtes à être transférées et il n'y avait plus de demandes *Publish* en file d'attente pour les transférer.

b) La minuterie d'édition d'une *Souscription* a expiré et il y avait soit des *Notifications* à envoyer, soit un *Message* d'entretien à envoyer.

Dans ces cas, la demande *Publish* nouvellement reçue est immédiatement traitée par la premiere *Souscription* qui rencontrera le cas a) ou le cas b).

Chaque fois que la minuterie d'édition expire, elle est immédiatement réinitialisée. S'il y a des *Notifications* ou un *Message* d'entretien à envoyer, il sort de la file d'attente une demande *Publish* et la traite. Lorsqu'une *Souscription* traite une demande *Publish*, il accède aux files d'attente de ses *MonitoredItems* et sort ses éventuelles *Notifications* de la file d'attente. Il retourne ces *Notifications* dans la réponse, positionnant le fanion *moreNotifications* s'il n'a pas été capable de retourner toutes les *Notifications* disponibles dans la réponse.

S'il y avait des *Notifications* ou un *Message* d'entretien à envoyer mais s'il n'y avait pas de demandes *Publish* en file d'attente, la *Souscription* suppose que la demande *Publish* est tardive et attend que la prochaine demande *Publish* soit reçue, comme décrit dans le cas b).

Si la *Souscription* est désactivée lorsque la minuterie d'édition expire ou s'il n'y a pas de *Notifications* disponibles, il entre dans l'état d'entretien et met le compteur d'entretien à sa valeur maximale comme défini pour la *Souscription*.

Pendant qu'il est dans l'état d'entretien, il recherche des *Notifications* chaque fois que la minuterie d'édition expire. Si une ou plusieurs notifications ont été générées, une demande *Publish* est sortie de la file d'attente et un *NotificationMessage* est retournée dans la réponse. En revanche, si la minuterie d'édition expire sans qu'une *Notification* ne devienne disponible, une demande *Publish* est sortie de la file d'attente et un *Message* d'entretien est retourné dans la réponse. La *Souscription* reviens alors dans son 'état normal qui est d'attendre que la minuterie d'édition expire de nouveau. Si, dans l'un ou l'autre de ces cas, il n'y a pas de demandes *Publish* dans la file d'attente, la *Souscription* attend que la prochaine demande *Publish* soit reçue, comme décrit dans le cas b).

Les états d'une *Souscription* sont définis au Tableau 79.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 79 – Etats d'un abonnement

État	Description
CLOSED	La *Souscription* n'a pas encore été créée ou s'est terminée.
CREATING	La *Souscription* est en cours de création.
NORMAL	La *Souscription* vérifie de façon cyclique s'il y a des *Notifications* issues de ses *MonitoredItems*. Le compteur d'entretien n'est pas utilisé dans cet état.
LATE	La minuterie d'édition a expiré et il y a des *Notifications* disponibles ou un *Message* d'entretien prêt à être envoyé mais il n'y a pas de demandes *Publish* en file d'attente. Dans cet état, la prochaine demande *Publish* est traitée lorsqu'elle est reçue. Le compteur d'entretien n'est pas utilisé dans cet état.
KEEPALIVE	La *Souscription* vérifie de façon cyclique s'il y a des *Notifications* issues de ses *MonitoredItems* ou si le compteur d'entretien compte à rebours jusqu'à 0 à partir de son maximum.

La table d'états est décrite au Tableau 80. Les règles et conventions suivantes s'appliquent:

a) Les *Evénements* représentent la réception de demandes de *Services* et l'occurrence d'*Evénements* internes, tels que les expirations de temporisateurs.

b) Les *Evénements* de demandes de *Services* peuvent être accompagnés de conditions qui soumettent à essai les valeurs des paramètres de *Service*. Les noms de paramètres commencent par une lettre minuscule.

c) Des *Evénements* internes peuvent être des conditions qui soumettent à essai les valeurs des *Variables* d'état. Les *Variables* d'état sont définies en 5.13.1.3. Elles commencent par une lettre majuscule.

d) Les *Evénements* internes et les demande de *Services* peuvent être accompagnés de conditions représentées par des fonctions dont la valeur de retour est soumise à essai. Les fonctions sont identifiées par "()" après leur nom. Elles sont décrites en 5.13.1.4.

e) Lorsqu'un *Evénement* est reçu, la première transition pour l'état courant est localisée et les transitions sont scrutées séquentiellement pour trouver la première transition qui satisfait aux critères de l'*Evénement* ou des conditions. Si aucune transition n'est trouvée, l'*Evénement* est ignoré.

f) Les actions sont décrites par des manipulations de fonctions et des *Variables* d'état.

g) L'*Evénement* LifetimeTimerExpires est déclenché lorsque son compteur associé atteint zéro.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 80 – Table d'états d'un abonnement

#	État courant	Evénement/Conditions	Action	Prochain état
1	CLOSED	Recevoir la demande CreateSubscription	CreateSubscription()	CREATING
2	CREATING	CreateSubscription échoue	ReturnNegativeResponse()	CLOSED
3	CREATING	CreateSubscription réussit	InitializeSubscription() MessageSent = FALSE ReturnResponse()	NORMAL
4	NORMAL	Recevoir la demande *Publish* && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && MoreNotifications == FALSE))	ResetLifetimeCounter() DeleteAckedNotificationMsgs() EnqueuePublishingReq()	NORMAL
5	NORMAL	Recevoir la demande *Publish* && PublishingEnabled == TRUE && MoreNotifications == TRUE	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnNotifications() MessageSent = TRUE	NORMAL
6	NORMAL	PublishingTimer expire && PublishingReqQueued == TRUE && PublishingEnabled == TRUE && NotificationsAvailable == TRUE	StartPublishingTimer() DequeuePublishReq() ReturnNotifications() MessageSent == TRUE	NORMAL
7	NORMAL	PublishingTimer expire && PublishingReqQueued == TRUE && MessageSent == FALSE && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() DequeuePublishReq() ReturnKeepAlive() MessageSent == TRUE	NORMAL
8	NORMAL	PublishingTimer expire && PublishingReqQueued == FALSE && (MessageSent == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == TRUE))	StartPublishingTimer()	LATE
9	NORMAL	PublishingTimer expire && MessageSent == TRUE && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() ResetKeepAliveCounter()	KEEPALIVE
10	LATE	Recevoir la demande *Publish* && PublishingEnabled == TRUE && (NotificationsAvailable == TRUE || MoreNotifications == TRUE)	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnNotifications() MessageSent = TRUE	NORMAL
11	LATE	Recevoir demande *Publish* && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE && MoreNotifications == FALSE))	ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnKeepAlive() MessageSent = TRUE	KEEPALIVE
12	LATE	PublishingTimer expire	StartPublishingTimer()	LATE
13	KEEPALIVE	Recevoir la demande *Publish*	ResetLifetimeCounter() DeleteAckedNotificationMsgs() EnqueuePublishingReq()	KEEPALIVE

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

#	État courant	Evénement/Conditions	Action	Prochain état
14	KEEPALIVE	PublishingTimer expire && PublishingEnabled == TRUE && NotificationsAvailable == TRUE && PublishingReqQueued == TRUE	StartPublishingTimer() DequeuePublishReq() ReturnNotifications() MessageSent == TRUE	NORMAL
15	KEEPALIVE	PublishingTimer expire && PublishingReqQueued == TRUE && KeepAliveCounter == 1 && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE)	StartPublishingTimer() DequeuePublishReq() ReturnKeepAlive() ResetKeepAliveCounter()	KEEPALIVE
16	KEEPALIVE	PublishingTimer expire && KeepAliveCounter > 1 && (PublishingEnabled == FALSE || (PublishingEnabled == TRUE && NotificationsAvailable == FALSE))	StartPublishingTimer() KeepAliveCounter--	KEEPALIVE
17	KEEPALIVE	PublishingTimer expire && PublishingReqQueued == FALSE && (KeepAliveCounter == 1 || (KeepAliveCounter > 1 && PublishingEnabled == TRUE && NotificationsAvailable == TRUE))	StartPublishingTimer()	LATE
18	NORMAL || LATE || KEEPALIVE	Recevoir la demande ModifySubscription	ResetLifetimeCounter() UpdateSubscriptionParams() ReturnResponse()	SAME
19	NORMAL || LATE || KEEPALIVE	Recevoir la demande SetPublishingMode	ResetLifetimeCounter() SetPublishingEnabled() MoreNotifications = FALSE ReturnResponse()	SAME
20	NORMAL || LATE || KEEPALIVE	Recevoir la demande Republish && RequestedMessageFound == TRUE	ResetLifetimeCounter() ReturnResponse()	SAME
21	NORMAL || LATE || KEEPALIVE	Recevoir la demande Republish && RequestedMessageFound == FALSE	ResetLifetimeCounter() ReturnNegativeResponse()	SAME
22	NORMAL || LATE || KEEPALIVE	Recevoir la demande TransferSubscriptions && SessionChanged() == FALSE	ResetLifetimeCounter() ReturnNegativeResponse ()	SAME
23	NORMAL || LATE || KEEPALIVE	Recevoir la demande TransferSubscriptions && SessionChanged() == TRUE && ClientValidated() ==TRUE	SetSession() ResetLifetimeCounter() DeleteAckedNotificationMsgs() ReturnResponse() IssueStatusChangeNotification()	SAME
24	NORMAL || LATE || KEEPALIVE	Recevoir la demande TransferSubscriptions && SessionChanged() == TRUE && ClientValidated() == FALSE	ReturnNegativeResponse()	SAME
25	NORMAL || LATE || KEEPALIVE	Recevoir la demande DeleteSubscriptions && SubscriptionAssignedToClient ==TRUE	DeleteMonitoredItems() DeleteClientPublReqQueue()	CLOSED

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

#	État courant	Evénement/Conditions	Action	Prochain état
26	NORMAL || LATE || KEEPALIVE	Recevoir la demande DeleteSubscriptions && SubscriptionAssignedToClient ==FALSE	ResetLifetimeCounter() ReturnNegativeResponse()	SAME
27	NORMAL || LATE || KEEPALIVE	LifetimeCounter == 1 Le LifetimeCounter est décrémenté si PublishingTimer expire et PublishingReqQueued == FALSE Le LifetimeCounter est réinitialisé si PublishingReqQueued == TRUE.	DeleteMonitoredItems() IssueStatusChangeNotification()	CLOSED

5.13.1.3 Variables d'état et paramètres

Les *Variables* d'état sont définies dans l'ordre alphabétique au Tableau 81.

Tableau 81 – Variables d'état et paramètres

Variable d'état	Description
MoreNotifications	Une valeur booléenne qui est mise à TRUE seulement par la fonction CreateNotificationMsg() lorsqu'il y avait de trop nombreuses *Notifications* pour un seul *NotificationMessage*.
LatePublishRequest	Une valeur booléenne qui est mise à TRUE pour refléter que, la dernière fois que la minuterie d'édition a expiré, il n'y avait pas de demandes *Publish* en file d'attente.
LifetimeCounter	Une valeur qui contient le nombre d'expirations consécutives de la minuterie d'édition sans activité du *Client* avant que la *Souscription* soit terminée.
MessageSent	Une valeur booléenne qui est mise à TRUE pour signifier que soit un *NotificationMessage,* soit un *Message* d'entretien a été envoyé sur la *Souscription*. Il s'agit d'un fanion qui est utilisé pour assurer qu'un *NotificationMessage* ou un *Message* d'entretien est envoyé la première fois que la minuterie d'édition expire.
NotificationsAvailable	Une valeur booléenne qui est mise à TRUE seulement lorsqu'il y a au moins un *MonitoredItem* qui est dans le mode production de rapport et qui a une *Notification* en file d'attente ou il y a au moins un élément à rapporter dont l'élément déclencheur a déclenché et a une *Notification* en file d'attente. La transition de cette *Variable* d'état de FALSE à TRUE crée l' *Evénement* «New *Notification* Queued» (Nouvelle notification placée en file d'attente) dans la table d'états.
PublishingEnabled	Le paramètre qui demande que l'édition soit activée ou désactivée.
PublishingReqQueued	Une valeur booléenne qui est mise à TRUE seulement lorsqu'il y a un *Message* de demande *Publish* mis en file d'attente de *Souscription*.
RequestedMessageFound	Une valeur booléenne qui est mise à TRUE seulement lorsque le *Message* dont la retransmission est demandée a été trouvé dans la file d'attente de retransmission.
SeqNum	La valeur qui enregistre la valeur du numéro de séquence utilisé dans les *NotificationMessages.*
SubscriptionAssignedToClient	Une valeur booléenne qui est mise à TRUE seulement lorsque la *Souscription* dont la suppression est demandée est affecté au *Client* qui a émis la demande. Une *Souscription* est affectée au *Client* qui l'a créée. Cette affectation peut seulement être changée par l'achèvement réussi du *Service* TransferSubscriptions.

5.13.1.4 Fonctions

Les fonctions d'action sont définies dans l'ordre alphabétique au Tableau 82.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 82 – Fonctions

État	Description
BindSession()	Lier la *Session Client* associée a une *Souscription* à la *Session Client* utilisée pour envoyer le *Service* traité. S'il s'agit de la derniere *Souscription* qui était lié au *Client*, effacer de la file d'attente des demandes *Publish* toutes les demandes *Publish* envoyées par le précédent *Client* et retourner des réponses négatives pour chacune.
ClientValidated()	Une fonction booléenne qui retourne TRUE seulement lorsque le *Client* qui présente une demande de TransferSubscriptions opère pour le compte du même utilisateur et prend en charge les mêmes *Profiles* que le *Client* de la *Session* précédente.
CreateNotificationMsg()	Incrémenter le SeqNum et créer un *NotificationMessage* à partir des *MonitoredItems* affectés a la *Souscription*. Sauvegarder dans la file d'attente de retransmission le *NotificationMessage* nouvellement créé. Si toutes les *Notifications* disponibles peuvent être envoyées dans la réponse *Publish*, la *Variable* d'état MoreNotifications est mise à FALSE. Autrement, elle est mise à TRUE.
CreateSubscription()	Tenter de créer une *Souscription*.
DeleteAckedNotificationMsgs()	Supprimer de la file d'attente de retransmission les *NotificationMessages* qui ont été acquittés par la demande.
DeleteClientPublReqQueue()	Vider la file d'attente des demandes *Publish* pour le *Client* qui envoie la demande DeleteSubscriptions, s'il n'y a plus de *Souscriptions* affectées au *Client* en question.
DeleteMonitoredItems()	Supprimer tous les *MonitoredItems* affectés à la *Souscription.*
DequeuePublishReq()	Sortir de la file d'attente une demande d'édition selon la règle du premier entré premier sorti. Valider si la demande d'édition est encore valide en vérifiant le timeoutHint dans le RequestHeader. Si la demande a dépassé le délai, envoyer un résultat de service *Bad_Timeout* pour la demande et sortir de la file d'attente une autre demande d'édition.
EnqueuePublishingReq()	Placer en file d'attente la demande d'édition.
InitializeSubscription()	ResetLifetimeCounter() MoreNotifications = FALSE PublishRateChange = FALSE PublishingEnabled = valeur du paramètre publishingEnabled dans la demande CreateSubscription PublishingReqQueued = FALSE SeqNum = 0 SetSession() StartPublishingTimer()
IssueStatusChangeNotification()	Émettre un *notificationMessage StatusChangeNotification* avec un code de statut pour le changement de statut d'une Subscription. Le type *StatusChangeNotification notificationMessage* est défini en 7.19.4. Le code de statut Bad_Timeout est utilisé si la durée de vie expire et Good_SubscriptionTransferred est utilisé si les Subscriptions étaient transférées à une autre Session.
ResetKeepAliveCounter()	Réinitialiser le compteur d'entretien au compte d'entretien maximal d'une *Souscription*. Le compte d'entretien maximal est établi par le *Client* lorsque la *Souscription* est créée et peut être modifié en utilisant le *Service* ModifySubscription.
ResetLifetimeCounter()	Réinitialiser la *Variable* LifetimeCounter à la valeur spécifiée pour la durée de vie d'une *Souscription* dans le *Service* CreateSubscription (5.13.2).
ReturnKeepAlive()	CreateKeepAliveMsg() ReturnResponse()
ReturnNegativeResponse ()	Retourner une réponse *Service* indiquant l'erreur de niveau *Service* appropriée. Il n'est retourné aucun paramètre autre que le responseHeader qui contient le *StatusCode* de niveau *Service*.

ReturnNotifications()	CreateNotificationMsg()
	ReturnResponse()
	If (MoreNotifications == TRUE) && (PublishingReqQueued == TRUE)
	{
	DequeuePublishReq()
	Boucler de nouveau à travers cette fonction.
	}

ReturnResponse()	Retourner la réponse appropriée, établissant les valeurs appropriées *StatusCodes* définis pour le *Service*.
SessionChanged()	Une fonction booléenne qui retourne TRUE seulement lorsque la *Session* utilisée pour envoyer une demande TransferSubscriptions est différente de la *Session Client* actuellement associée à la *Souscription*.
SetPublishingEnabled ()	Mettre la *Variable* d'état PublishingEnabled à la valeur du paramètre publishingEnabled reçu dans la demande.
SetSession	Établir les informations de *Session* pour que la *Souscription* corresponde à la *Session* sur laquelle la demande TransferSubscriptions a été émise.
StartPublishingTimer()	Démarrer ou redémarrer la minuterie d'édition et décrémenter la *Variable* LifetimeCounter.
UpdateSubscriptionParams()	Négocier et mettre à jour les paramètres d'une*Souscription*. Si le nouvel intervalle d'entretien est inférieur à la valeur courante du compteur d'entretien, exécuter les fonctions ResetKeepAliveCounter() et ResetLifetimeCounter().

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.2 Créer une souscription (CreateSubscription)

5.13.2.1 Description

Ce *Service* est utilisé pour créer une *Souscription*. Les *Souscriptions* surveillent un jeu de *MonitoredItems* pour détecter des *Notifications* et les retourner au *Client* en réponse aux demandes *Publish*.

5.13.2.2 Paramètres

Le Tableau 83 définit les paramètres du *Service*.

Tableau 83 – Paramètres du service CreateSubscription

Nom	Type	Description
Request		
requestHeader	Request Header	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
requestedPublishing Interval	Duration	Cet intervalle définit la fréquence cyclique à laquelle il est demandé à une *Souscription* de retourner des *Notifications* au *Client*. Cet intervalle est exprimé en millisecondes. Cet intervalle est représenté par la minuterie d'édition dans la table d'états de la *Souscription* (voir 5.13.1.2). La valeur négociée pour ce paramètre retournée dans la réponse est utilisée comme intervalle d'échantillonnage par défaut pour les *MonitoredItems* affectés à cette *Souscription*. La valeur 0 n'est pas valide.
requestedLifetimeCount	Counter	Compte de durée de vie demandé (voir 7.5 pour la définition de *Counter*). Le compte de durée de vie doit être au minimum égal à trois fois le compte d'entretien. Lorsque la minuterie d'édition a expiré ce nombre de fois sans qu'une demande *Publish* soit disponible pour envoyer un *NotificationMessage*, la *Souscription* doit être supprimée par le *Serveur*.
requestedMaxKeepAlive Count	Counter	Compte d'entretien maximal demandé (voir 7.5 pour la définition de compteur *Counter*). Lorsque la minuterie d'édition a expiré ce nombre de fois sans exiger qu'un moindre *NotificationMessage* soit envoyé, la *Souscription* envoie un *Message* d'entretien au *Client*. La valeur 0 n'est pas valide.
maxNotificationsPerPublish	Counter	Le nombre maximal des notifications que le *Client* souhaite recevoir dans une seule réponse *Publish*. Une valeur de zéro indique qu'il n'y a pas de limite.
publishingEnabled	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE l'édition est activée pour la *Souscription.* FALSE l'édition est désactivée pour la *Souscription*. La valeur de ce paramètre n'altère pas la valeur de l' *Attribut* de mode de surveillance des *MonitoredItems*.
priority	Byte	Indique la priorité relative d'une *Souscription*. Lorsqu'il est nécessaire que plus d'une *Souscription* envoient des *Notifications*, il convient que le *Serveur* sorte de la file d'attente une demande Publish adressée à la *Souscription* ayant le nombre *priority* le plus élevé. Pour les *Souscriptions* d'égale *priority*, il convient que le *Serveur* sorte de la file d'attente les demandes Publish en «round-robin» (chacune à tour de rôle). Lorsque la période d'entretien expire pour une *Souscription*, elle devient prioritaire quelle que soit son niveau de priorité, afin d'éviter que la *Souscription* n'expire. Il convient qu'un Client qui n'exige pas de réglages de priorité spéciaux mette cette valeur à zéro.
Response		
responseHeader	Response Header	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* (voir 7.13 pour la définition d'*IntegerId*). Cet identificateur doit être unique pour le *Serveur* tout entier, pas uniquement pour la *Session*, afin de permettre le transfert d'une *Souscription* vers une autre *Session* au moyen du service TransferSubscriptions.
revisedPublishingInterval	Duration	L'intervalle d'édition réel que le *Serveur* utilisera, exprimé en millisecondes. Il convient que le *Serveur* essaie d'honorer la demande du *Client* pour ce paramètre, mais il peut négocier cette valeur à la hausse ou à la baisse pour satisfaire à ses propres contraintes.
revisedLifetimeCount	Counter	La durée de vie d'une *Souscription* doit être au moins égale à trois fois l'intervalle d'entretien négocié avec le *Serveur.*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
Request		
revisedMaxKeepAliveCount	Counter	Le compte d'entretien maximal réel. (voir 7.5 pour la définition de *Counter*). Il convient que le *Serveur* essaie d'honorer la demande du *Client* pour ce paramètre, mais il peut négocier cette valeur à la hausse ou à la baisse pour satisfaire à ses propres contraintes.

5.13.2.3 Résultats des services

Le Tableau 84 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 84 – Codes de résultats des services CreateSubscription

Id symbolique	Description
Bad_TooManySubscriptions	Le *Serveur* a atteint le nombre maximal d'abonnements.

5.13.3 Modifier une souscription (ModifySubscription)

5.13.3.1 Description

Ce *Service* est utilisé pour modifier une *Souscription*.

5.13.3.2 Paramètres

Le Tableau 85 définit les paramètres du *Service*. Les modifications apportées à l'intervalle d'édition prendront effet la prochaine fois que la minuterie d'édition expirera.

Tableau 85 – Paramètres du service ModifySubscription

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* (voir 7.13 pour la définition d'*IntegerId*).
requestedPublishingInterval	Duration	Cet intervalle définit la fréquence cyclique à laquelle il est demandé à la *Souscription* de retourner des *Notifications* au *Client*. Cet intervalle est exprimé en millisecondes. Cet intervalle est représenté par la minuterie d'édition dans la table d'états d'une *Souscription* (voir 5.13.1.2). La valeur négociée pour ce paramètre retournée dans la réponse est utilisée comme intervalle d'échantillonnage par défaut pour les *MonitoredItems* affectés à cette *Souscription*. La valeur 0 n'est pas valide.
requestedLifetimeCount	Counter	Compte de durée de vie demandé (voir 7.5 pour la définition de *Counter*). Le compte de durée de vie doit être au minimum égal à trois fois le compte d'entretien. Lorsque la minuterie d'édition a expiré ce nombre de fois sans qu'une demande *Publish* soit disponible pour envoyer un *NotificationMessage*, la *Souscription* doit être supprimé par le *Serveur*.
requestedMaxKeepAliveCount	Counter	Le compte d'entretien maximal demandé. (voir 7.5 pour la définition de *Counter*). Lorsque la minuterie d'édition a expiré ce nombre de fois sans exiger qu'un moindre *NotificationMessage* soit envoyé, la *Souscription* envoie un *Message* d'entretien au *Client*. La valeur 0 n'est pas valide.
maxNotificationsPerPublish	Counter	Le nombre maximal des notifications que le *Client* souhaite recevoir dans une seule réponse *Publish*. Une valeur de zéro indique qu'il n'y a pas de limite.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
Request		
Priority	Byte	Indique la priorité relative d'une *Souscription*. Lorsqu'il est nécessaire que plus d'une *Souscription* envoient des *Notifications*, il convient que le *Serveur* sorte de la file d'attente une demande Publish adressée à la *Souscription* ayant le nombre *priority* le plus élevé. Pour les *Souscriptions* d'égale *priority*, il convient que le *Serveur* sorte de la file d'attente les demandes Publish en «round-robin» (chacune à tour de rôle). Toute *Souscription* qui a besoin d'envoyer un *Message* d'entretien doit avoir la préséance quelle que soit sa *priority*, afin d'éviter que la *Souscription* n'expire. Il convient qu'un Client qui n'exige pas de réglages de priorité spéciaux mette cette valeur à zéro.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
revisedPublishingInterval	Duration	L'intervalle d'édition réel que le *Serveur* utilisera, exprimé en millisecondes. Il convient que le *Serveur* essaie d'honorer la demande du *Client* pour ce paramètre, mais il peut négocier cette valeur à la hausse ou à la baisse pour satisfaire à ses propres contraintes.
revisedLifetimeCount	Counter	La durée de vie d'une *Souscription* doit être au moins égale à trois fois l'intervalle d'entretien négocié avec le *Serveur.*
revisedMaxKeepAliveCount	Counter	Le compte d'entretien maximal réel (voir 7.5 pour la définition de *Counter*). Il convient que le *Serveur* essaie d'honorer la demande du *Client* pour ce paramètre, mais il peut négocier cette valeur à la hausse ou à la baisse pour satisfaire à ses propres contraintes.

5.13.3.3 Résultats des services

Le Tableau 86 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 86 – Codes de résultats des services ModifySubscription

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.13.4 Demander un mode de publication (SetPublishingMode)

5.13.4.1 Description

Ce *Service* est utilisé pour permettre l'envoi de *Notifications* sur une ou plusieurs *Souscriptions*.

5.13.4.2 Paramètres

Le Tableau 87 définit les paramètres du *Service*.

Tableau 87 – Paramètres du service SetPublishingMode

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
publishingEnabled	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE l'édition de *NotificationMessages* est activée pour la *Souscription*. FALSE l'édition de *NotificationMessages* est désactivée pour la *Souscription*. La valeur de ce paramètre n'altère pas la valeur de l'*Attribut* de mode de surveillance des *MonitoredItems*. La mise de cette valeur à FALSE n'interrompt pas l'envoi de *Messages* d'entretien.
subscriptionIds []	IntegerId	Liste d'identificateurs affectés par le *Serveur* pour les *Souscriptions* à activer ou désactiver (voir 7.13 pour la définition d'*IntegerId*).
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

results []	StatusCode	Liste des *StatusCodes* pour les *Souscriptions* à activer/désactiver (voir 7.33 pour la définition des *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *Souscriptions* à activer/désactiver (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.13.4.3 Résultats des services

Le Tableau 88 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 88 – Codes de résultats des services SetPublishingMode

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.13.4.4 StatusCodes

Le Tableau 89 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 89 – Codes de résultats de niveau opération de SetPublishingMode

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

5.13.5 Publier (Publish)

5.13.5.1 Description

Ce *Service* est utilisé à deux fins. Premièrement, il est utilisé pour acquitter réception des *NotificationMessages* pour une ou plusieurs *Souscriptions*. Deuxièmement, il est utilisé pour demander au *Serveur* de retourner un *NotificationMessage* ou un *Message* d'entretien. Comme les demandes *Publish* ne sont pas dirigées à une *Souscription* spécifique, elles peuvent être utilisées par n'importe quelle *Souscription*. Le paragraphe 5.13.1.2 décrit l'utilisation du *Service Publish*.

Les stratégies du *Client* pour émettre des demandes *Publish* peuvent varier en fonction des retards réseaux entre le *Client* et le *Serveur*. Dans de nombreux cas, le *Client* peut souhaiter émettre une demande *Publish* immédiatement après avoir créé une *Souscription* et, par la suite, immédiatement après avoir reçu une réponse *Publish*.

Dans d'autres cas, notamment dans les réseaux à temps d'attente élevée, le *Client* peut souhaiter placer en cascade les demandes *Publish* pour assurer une production cyclique de rapports par le *Serveur*. Le placement en cascade implique d'envoyer plus d'une demande *Publish* pour chaque *Souscription* avant de recevoir une réponse. Par exemple, si le réseau introduit entre le *Client* et le *Serveur* un retard de 5 s et l'intervalle d'édition pour une *Souscription* est 1 s le *Client* aura à émettre des demandes *Publish* toutes les secondes au lieu d'attendre qu'une réponse soit reçue avant d'envoyer la prochaine demande.

Il convient qu'un *Serveur* limite le nombre de demandes *Publish* actives afin d'éviter que le nombre soit infini car les demandes *Publish* sont censées être placées en file d'attente dans le *Serveur*. Mais un *Serveur* doit accepter plus de demandes *Publish* mises en file d'attente que d'Abonnements créés. Il est attendu qu'un *Serveur* prenne en charge plusieurs demandes *Publish* par *Souscription*. Lorsqu'un *Serveur* reçoit une nouvelle demande *Publish* qui dépasse sa limite, il doit sortir de la file d'attente la demande *Publish* la plus ancienne et retourner une réponse avec le résultat mis à *Bad_TooManyPublishRequests*. Si le *Client*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

reçoit ce résultat pour une demande *Publish*, il ne doit pas émettre d'autre demande *Publish* avant que l'une de ses demandes *Publish* en cours ne soit retournée du *Serveur*.

Les *Clients* peuvent limiter la taille des réponses *Publish* avec le paramètre *maxNotificationsPerPublish* passé au *Service CreateSubscription*. Cependant, cela pourrait encore produire un message à traiter trop volumineux pour le *Client* ou le *Serveur*. Dans cette situation, le *Client* s'apercevra soit que le *Canal Sécurisé* passe à un état de défaut et a besoin d'être rétablie, soit que la réponse *Publish* retourne une erreur et l'appel du *Service Republish* retourne aussi une erreur. Si l'une de ces situations se produit, le *Client* aura à ajuster ses limites de traitement de message pour la *Souscription* et/ou les *MonitoredItems*.

Le réglage des informations de diagnostic de retour diagnostic dans l'en-tête de demande des *CreateMonitoredItems* ou le dernier *Service ModifyMonitoredItems* est appliqué aux *Monitored Items* (Eléments surveillés) et sert de réglages d'informations de diagnostic lors de l'envoi de Notifications dans la réponse *Publish*.

5.13.5.2 Paramètres

Le Tableau 90 définit les paramètres du *Service*.

Tableau 90 – Paramètres du service Publish

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscription Acknowledgements []	Subscription Acknowledgement	La liste des acquittements pour une ou plusieurs *Souscriptions*. Cette liste peut contenir plusieurs acquittements pour la même *Souscription* (plusieurs entrées ayant le même *subscriptionId*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour une *Souscription* (voir 7.13 pour la définition d'*IntegerId*).
sequenceNumber	Counter	Le numéro de séquence acquitté (voir 7.5 pour la définition de *Counter*). Le *Serveur* peut supprimer de sa file d'attente de retransmission le *Message* ayant ce numéro de séquence.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* pour lequel les *Notifications* sont retournées (voir 7.13 pour la définition d'*IntegerId)*. La valeur 0 est utilisée pour indiquer qu'il n'y avait pas de *Souscriptions* définis pour lesquelles une réponse pouvait être envoyée.
availableSequence Numbers []	Counter	Une liste de plages de numéros de séquence qui identifient des *NotificationMessages* non acquittés qui sont disponibles pour la retransmission à partir de la file d'attente de retransmission d'une *Souscription*. Cette liste est élaborée après traitement des acquittements dans la réponse (voir 7.5 pour la définition de *Counter*).
moreNotifications	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE le nombre des *Notifications* qui étaient prêtes à être envoyées n'a pas pu être envoyé dans une seule réponse. FALSE toutes les *Notifications* qui étaient prêtes sont incluses dans la réponse.
notificationMessage	Notification Message	Le *NotificationMessage* qui contient la liste des *Notifications*. Le type de paramètre *NotificationMessage* est spécifié en 7.20.
results []	StatusCode	Liste de résultats pour les acquittements (voir 7.33 pour la définition de *StatusCode*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionAcknowledgements*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les acquittements (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionAcknowledgements*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.5.3 Résultats des services

Le Tableau 91 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 91 – Codes de résultats des services Publish

Id symbolique	Description
Bad_TooManyPublishRequests	Le serveur a atteint le nombre maximal de demandes d'édition placées en file d'attente.
Bad_NoSubscription	Aucun abonnement n'est disponible pour cette session.

5.13.5.4 StatusCodes

Le Tableau 92 définit les valeurs pour le paramètre *acknowledgeResults* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 92 – Codes de résultats de niveau opération de Publish

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_SequenceNumberUnknown	Le numéro de séquence est inconnu du serveur.

5.13.6 Publier à nouveau (Republish)

5.13.6.1 Description

Ce *Service* demande a une *Souscription* de rééditer un *NotificationMessage* à partir de sa file d'attente de retransmission. Si le *Serveur* n'a pas le *Message* demandé dans sa file d'attente de retransmission, il retourne une réponse d'erreur.

Voir 5.13.1.2 pour a description détaillée du comportement de ce *Service*.

5.13.6.2 Paramètres

Le Tableau 93 définit les paramètres du *Service*.

Tableau 93 – Paramètres du service Republish

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionId	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* à rééditer (voir 7.13 pour la définition d'*IntegerId*).
retransmitSequence Number	Counter	Le numéro de séquence d'un *NotificationMessage* spécifique à rééditer (voir 7.5 pour la définition de *Counter*).
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
notificationMessage	Notification Message	Le *NotificationMessage* demandé. Le type de paramètre *NotificationMessage* est spécifié en 7.18.

5.13.6.3 Résultats des services

Le Tableau 94 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 94 – Codes de résultats des services Republish

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_MessageNotAvailable	Le message demandé n'est plus disponible.

5.13.7 Transferer une souscription (TransferSubscriptions)

5.13.7.1 Description

Ce *Service* est utilisé pour transférer une *Souscription* et ses *MonitoredItems* d'une *Session* à l'autre. Par exemple, un *Client* peut avoir besoin de rouvrir une *Session* et ensuite de transférer ses *Souscriptions* à la *Session* en question. Il peut aussi être utilisé par un *Client* pour prendre une *Souscription* à un autre *Client* en transférant la *Souscription* à sa *Session*.

L'*authenticationToken* contenu dans l'en-tête de demande identifie la *Session* à laquelle la *Souscription* et les *MonitoredItems* doivent être transférés. Le *Serveur* doit valider que le *Client* de la *Session* en question opère pour le compte du même utilisateur et que le *Client* potentiellement nouveau prend en charge les *Profiles* qui sont nécessaires pour la *Souscription*. Si le *Serveur* transfère la *Souscription*, il retourne les numéros de séquence des *NotificationMessages* qui sont disponibles pour une retransmission. Il convient que le *Client* acquitte tous les *Messages* de la liste pour lesquels il ne demandera pas de retransmission.

Si le *Serveur* transfère la *Souscription* à la nouvelle *Session*, le *Serveur* doit émettre un *notificationMessage StatusChangeNotification* avec le code de statut Good_SubscriptionTransferred. Le type *notificationMessage StatusChangeNotification* est défini en 7.19.4.

5.13.7.2 Paramètres

Le Tableau 95 définit les paramètres du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 95 – Paramètres du service TransferSubscriptions

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionIds []	IntegerId	Liste d'identificateurs pour les *Souscriptions* à transférer au nouveau *Client* (voir 7.13 pour la définition d'*IntegerId)*. Ces identificateurs sont transférés du *Client* principal vers un *Client* de secours via des mécanismes externes.
sendInitialValues	Boolean	Un paramètre *Boolean* ayant les valeurs suivantes: TRUE la première réponse Publish après l'appel de TransferSubscriptions doit contenir les valeurs courantes de tous les Monitored Items (éléments surveillés) dans l'Abonnement dans lequel le Mode "Monitoring" est mis à "Reporting". FALSE la première réponse Publish après l'appel de TransferSubscriptions doit seulement contenir les changements de valeur depuis que la dernière réponse Publish a été envoyée. Ce paramètre ne s'applique qu'aux MonitoredItems utilisés pour surveiller les modifications d'Attribut.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	TransferResult	Liste de résultats pour les *Souscriptions* à transférer. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*.
statusCode	StatusCode	*StatusCode* pour chaque *Souscription* à transférer (voir 7.33 pour la définition de *StatusCode*).
availableSequence NumbersRanges []	Counter	Une liste de plages de numéros de séquence qui identifie des *NotificationMessages* qui sont dans la file d'attente de retransmission d'une *Souscription*. Ce paramètre est null si le transfert d'une *Souscription* a échoué. Le type *NumericRange* est défini en 7.21.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *Souscriptions* à transférer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.13.7.3 Résultats des services

Le Tableau 96 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 96 – Codes de résultats des services TransferSubscriptions

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_UserAccessDenied	Voir le Tableau 165 pour la description de ce code de résultat. Le *Client* de la *Session* courante n'opère pas pour le compte du même utilisateur que la *Session* qui est propriétaire d'une *Souscription*.
Bad_InsufficientClientProfile	Le *Client* de la *Session* courante ne prend pas en charge un ou plus *Profiles* qui sont nécessaires a une *Souscription*.

5.13.7.4 StatusCodes

Le Tableau 97 définit les valeurs de niveau opération pour le paramètre *statusCode* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 97 – Codes de résultats de niveau opération de TransferSubscriptions

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.13.8 Supprimer des souscriptions (DeleteSubscriptions)

5.13.8.1 Description

Ce *Service* est invoqué par le *Client* pour supprimer une ou plusieurs *Souscriptions* qu'il a créées et qui n'ont pas été transférées à un autre *Client* ou qui lui ont été transférées.

L'achèvement réussi de ce *Service* entraîne la suppression de tous les *MonitoredItems* qui utilisent la *Souscription*. S'il s'agit d'une derniere *Souscription* affectée au *Client* qui émet la demande, toutes les demandes *Publish* placées en file d'attente par le *Client* en question sont sorties de la file d'attente et une réponse négative est retournée pour chacune de ces demandes.

5.13.8.2 Paramètres

Le Tableau 98 définit les paramètres du *Service*.

Tableau 98 – Paramètres du service DeleteSubscriptions

Nom	Type	Description
Request		
requestHeader	RequestHeader	Paramètres communs aux demandes (voir 7.26 pour la définition de *RequestHeader*).
subscriptionIds []	IntegerId	L'identificateur affecté par le *Serveur* pour la *Souscription* ((voir 7.13 pour la définition d'*IntegerId)*.
Response		
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).
results []	StatusCode	Liste des *StatusCodes* pour les *Souscriptions* à supprimer (voir 7.33 pour la définition de *StatusCode)*. La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic pour les *Souscriptions* à supprimer (voir 7.8 pour la définition de *DiagnosticInfo*). La taille et l'ordre de la liste correspondent à la taille et à l'ordre du paramètre de demande *subscriptionIds*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande.

5.13.8.3 Résultats des services

Le Tableau 99 définit les résultats des *Services* spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 165.

Tableau 99 – Codes de résultats des services DeleteSubscriptions

Id symbolique	Description
Bad_NothingToDo	Voir le Tableau 165 pour la description de ce code de résultat.
Bad_TooManyOperations	Voir le Tableau 165 pour la description de ce code de résultat.

5.13.8.4 StatusCodes

Le Tableau 100 définit les valeurs pour le paramètre *results* qui sont spécifiques à ce *Service*. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 100 – Codes de résultats de niveau opération de DeleteSubscriptions

Id symbolique	Description
Bad_SubscriptionIdInvalid	Voir le Tableau 165 pour la description de ce code de résultat.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6 Comportements des services

6.1 Sécurité

6.1.1 Vue générale

Les *Services* OPC UA définissent un certain nombre de mécanismes pour satisfaire aux exigences de sécurité décrites dans les grandes lignes dans la CEI/TR 62541-2. Ce paragraphe décrit un certain nombre d'importantes procédures de sécurité que les *Applications OPC UA* doivent suivre.

6.1.2 Obtenir et installer un certificat d'instance d'application (Application Instance Certificate)

Toutes les *Applications OPC UA* exigent un certificat d'instance d'application qui doit contenir les renseignements suivants:

- le nom du réseau ou l'adresse de l'ordinateur sur lequel l'application s'exécute;
- le nom de l'organisation qui gère ou est le propriétaire de l'application;
- le nom de l'application;
- l'URI de l'instance d'application;
- le nom de l'autorité de certification qui a émis le certificat;
- la date d'émission et d'expiration du certificat;
- la clé publique émise pour l'application par l'autorité de certification (CA, certificate authority);
- une signature numérique créée par l'autorité de certification (CA).

En outre, chaque certificat d'instance d'application a une clé privée qu'il convient de conserver dans un endroit uniquement accessible par l'application. Si l'intégrité de cette clé privée est compromise, l'administrateur doit affecter un nouveau certificat d'instance d'application et une nouvelle clé privée à l'application.

Ce certificat peut être généré automatiquement lorsque l'application est installée. Dans cette situation, la clé privée affectée au certificat doit être utilisée pour créer la signature du certificat. Les certificats créés de cette manière sont appelés certificats autosignés.

Si l'administrateur responsable de l'application décide qu'un certificat auto-signé ne satisfait aux exigences de sécurité de l'organisation, il convient que l'administrateur installe un certificat émis par une autorité de certification. Les étapes impliquées dans la demande de certificat d'instance d'application auprès d'une autorité de certification sont montrées à la Figure 18.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

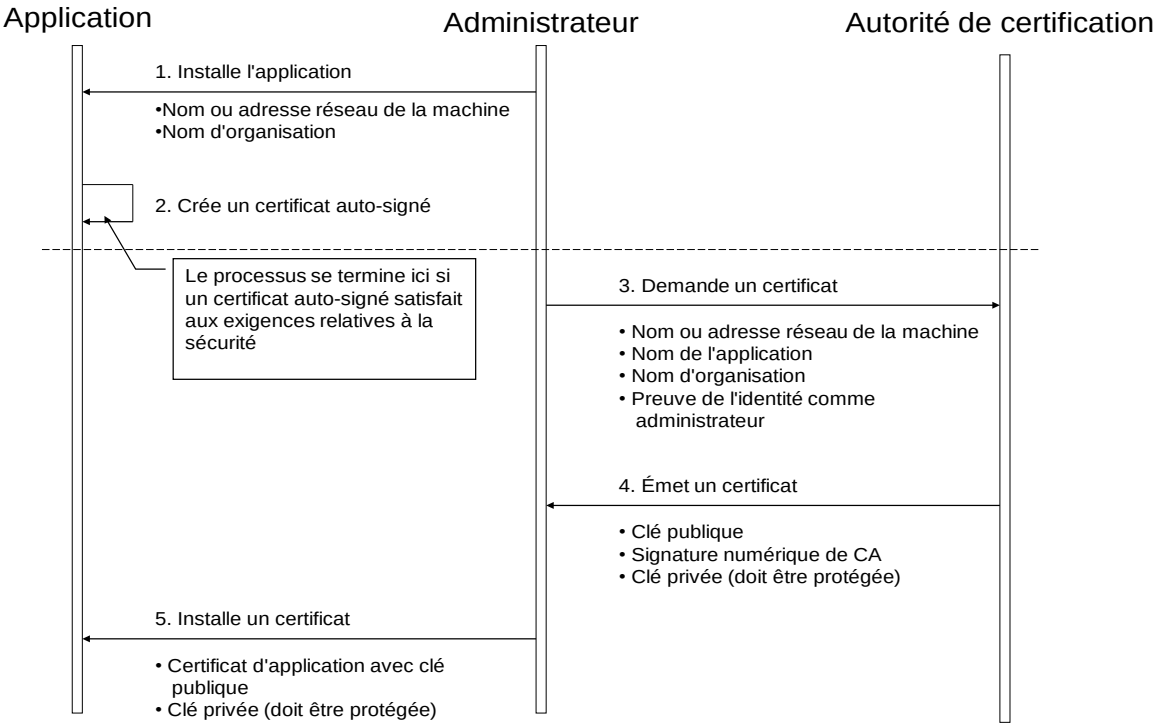


Figure 18 – Obtenir et installer un certificat d'instance d'application

La Figure 18 illustre les interactions entre l'*Application*, l'*Administrateur* et le *CertificateAuthority*. L'*Application* est en tant qu'*Application OPC UA* installée sur une seule machine. L'*Administrateur* est la personne chargée de gérer la machine et *l'Application OPC UA*. Le *CertificateAuthority* est une entité qui peut émettre des certificats numériques qui satisfont aux exigences de l'organisation qui déploie l' *Application OPC UA*.

Si l'*Administrateur* décide qu'un certificat auto-signé satisfait aux exigences de sécurité pour l'organisation, l'*Administrateur* peut sauter les Étapes 3 à 5. Les vendeurs d'application doivent toujours créer un certificat auto-signé par défaut au cours du processus d'installation. Chaque *Application OPC UA* doit permettre aux *Administrateurs* de remplacer des certificats d'instance d'application par des certificats qui satisfont à leurs exigences.

Lorsque l'*Administrateur* demande un nouveau certificat auprès d'une autorité de certification, l'autorité de certification peut exiger que l'*Administrateur* fournisse la preuve de l'autorisation de demander des certificats pour l'organisation qui sera propriétaire du certificat. Le mécanisme exact utilisé pour fournir cette preuve dépend de l'autorité de certification.

Les vendeurs peuvent choisir d'automatiser le processus d'acquisition de certificats auprès d'une autorité. Si tel est le cas, l'*Administrateur* passe toujours par les étapes illustrées de la Figure 18. En revanche, le programme d'installation pour l'application le fait automatiquement et sollicite seulement l'*Administrateur* pour qu'il fournisse les renseignements relatifs à l'instance d'application installée.

6.1.3 Obtenir et installer un Certificat de logiciel

Toutes les *Applications OPC UA* peuvent avoir un ou plusieurs certificats de logiciel qui sont émis par des autorités de certification et spécifient les profils que l'application prend en charge. Ces certificats de logiciel contiennent les informations suivantes:

- le nom du vendeur responsable du produit;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

le nom du produit;
l'URI du produit;
le numéro de version et d'implémentation du logiciel;
le certificat du produit du vendeur;
une liste des profils pris en charge par l'application;
le statut d'essais de certification pour chaque profil pris en charge;
le nom de l'autorité de controle qui a émis le certificat;
la date d'émission et d'expiration du certificat;
la clé publique émise pour l'application par l'autorité de certification;
une signature numérique créée par l'autorité de certification.

Le vendeur du produit a la responsabilité de mener à bien le processus de certification et de demander des certificats de logiciel auprès des autorités de certification. Le vendeur doit installer les certificats des logiciels lorsqu'une application est installée sur une machine. Lorsqu'ils distribuent ces certificats, il convient que les vendeurs d'application prennent des précautions pour empêcher que des utilisateurs non autorisés acquièrent leurs certificats logiciels et les utilisent pour des applications que le vendeur n'a pas développées. Les certificats de logiciels improprement utilisés ne sont pas un risque pour la sécurité, mais les vendeurs pourraient être réprimandés pour des problèmes d'interopérabilité dus à l'utilisation de certificats par des applications non autorisées.

Des vendeurs peuvent choisir d'affecter leur propre *Certificat* à l'application (appelé certificat produit du vendeur). Ce certificat serait alors incorporé dans le *Software Certificate*. La clé privée pour le certificat produit du vendeur est gérée par le Vendeur.

Les étapes impliquées dans l'acquisition et l'installation d'un certificat de logiciel auprès d'une autorité de certification sont montrées à la Figure 19.

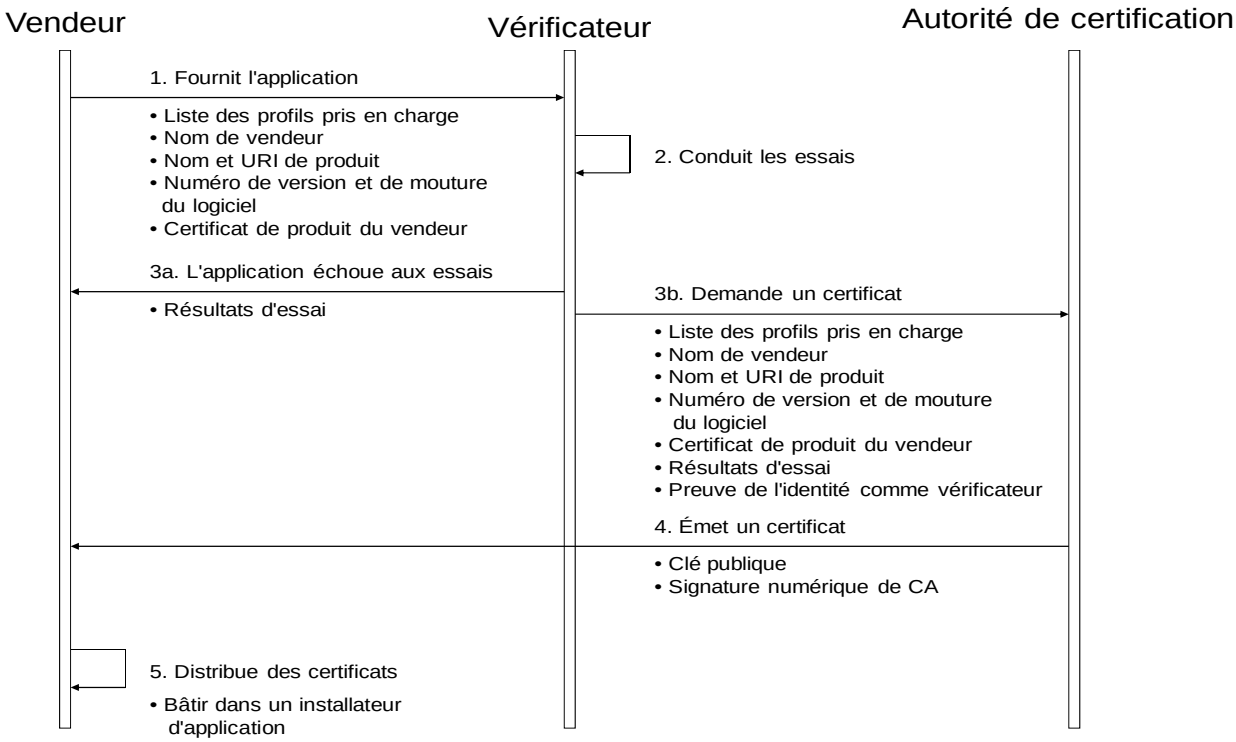


Figure 19 – Obtention et installation d'un Certificat de logiciel.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

La Figure 19 illustre les interactions entre le *Vendeur (Vendor)*, le *Vérificateur (Tester)* et la *CertifyingAuthority*. Le *Vendeur* est l'organisation responsable de l'*Application OPC UA*. Le *Vérificateur* peut être le vendeur (pour l'auto-certification) ou peut être un moyen d'essais de tierce partie. La *CertifyingAuthority* est une Autorité de certification ICP (infrastructure à clé publique) gérée par l'organisation qui a créé les profils d'*Application OPC UA* et les programmes de certification.

La *CertifyingAuthority* émettra des certificats uniquement pour les personnes en qui elle a confiance. Pour cette raison, le *Vérificateur* doit fournir une preuve d'identité avant que la *CertifyingAuthority* n'émette un certificat.

6.1.4 Déterminer si un certificat est fiable

Les *Applications* ne doivent jamais communiquer avec une autre application en laquelle elles n'ont pas confiance. Une *Application* décide qu'une autre application est fiable en vérifiant si le *Certificat* d'instance d'application de l'autre application est fiable. Les applications doivent s'appuyer sur les listes de *Certificat*s fourni par l'Administrateur pour déterminer la confiance. Il existe deux listes distinctes: Une liste d'Applications fiables et une liste d'Autorités de certification (CA) fiables. Si une application n'est pas directement fiable (c'est-à-dire que son certificat ne figure pas dans la liste d'Applications fiables), l'Application doit construire une chaîne de *Certificat*s ramenant à une CA fiable.

Lors de la construction d'une chaîne, chaque *Certificat* de la chaîne doit être validé. S'il se produit une quelconque erreur de validation, la vérification de la confiance a échoué. Certaines erreurs de validation ne sont pas critiques, ce qui signifie qu'elles peuvent être éliminées par l'utilisateur d'une Application ayant les privilèges appropriés. Les erreurs de validation supprimées sont toujours consignées dans un rapport via un audit (cela signifie qu'un événement d'Audit approprié est soulevé).

La construction d'une chaîne de confiance nécessite l'accès à tous les *Certificat*s de la chaîne. Ces *Certificat*s peuvent être stockés localement ou ils peuvent être fournis avec le *Certificat* d'application. Le traitement s'arrête si un certificat de CA ne peut pas être trouvé ou s'il est dans la liste des CA fiables.

Le Tableau 101 spécifie les étapes utilisées pour valider un *Certificat* et l'ordre dans lequel il convient de les suivre. Ces étapes sont répétées pour chaque certificat jusqu'à ce qu'un certificat fiable ait été trouvé. Chaque étape de validation a un statut d'erreur et un type d'événement d'audit qui sont uniques et doivent être consignés dans un rapport si la vérification échoue. L'événement d'audit s'ajoute à tout événement d'audit qui avait été généré pour le *Service* particulier qui avait été invoqué. L'événement d'audit du *Service* dans le texte de son message doit inclure l'eventID d'audit de l'*AuditCertifcateEvent* (pour plus de détails, voir 6.2). Le traitement s'arrête si une erreur se produit, à moins qu'elle ne soit pas critique et qu'elle ait été éliminée.

Les certificats d'instance d'*Application* ne doivent pas être utilisés dans un *Client* ou un *Serveur* jusqu'à ce qu'ils soient évalués et marqués comme fiables. Cela peut se produire de manière automatique par une chaîne de confiance ICP ou de manière hors ligne où le *Certificat* est marqué comme étant fiable par un administrateur après évaluation.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 101 – Étapes de validation d'un certificat

Étape	Erreur/Événement d'audit	Description
Structure du certificat	Bad_SecurityChecksFailed AuditCertificateInvalidEventType	La structure du certificat est vérifiée. Cette erreur ne peut pas être supprimée.
Période de validité	Bad_CertificateTimeInvalid Bad_CertificateIssuerTimeInvalid AuditCertificateExpiredEventType	L'instant actuel doit se situer après le début de la période de validité et avant la fin. Cette erreur peut être supprimée.
Nom de l'hôte	Bad_CertificateHostNameInvalid AuditCertificateDataMismatchEventType	Le HostName dans l'URL utilisé pour se connecter au *Serveur* doit être le même que l'un des HostNames spécifiés dans le Certificat. Cette erreur peut être supprimée.
URI	Bad_CertificateUriInvalid AuditCertificateDataMismatchEventType	Les Certificats d'Application et de Logiciel contiennent un URI d'application ou de produit qui doit correspondre à l'URI spécifié dans l'*ApplicationDescription* fournie avec le produit. Cette vérification est ignorée pour les Certificats de CA. Cette erreur ne peut pas être supprimée. Le *gatewayServerUri* est utilisé pour valider un *Application Certificate* lors de la connexion à un *Gateway Server* (voir 7.1).
Utilisation du certificat	Bad_CertificateUseNotAllowed Bad_CertificateIssuerUseNotAllowed AuditCertificateMismatchEventType	Chaque Certificat a un ensemble d'utilisations pour le Certificat. Ces utilisations doivent correspondre à l'utilisation demandée pour le Certificat (à savoir Application, Logiciel ou CA). Cette erreur peut être supprimée à moins que le Certificat n'indique que l'utilisation est obligatoire.
Vérification de la liste de confiance	None	Aucune autre vérification n'est requise si le Certificat figure dans la Liste de confiance. L'Administrateur doit valider complètement tout Certificat avant de le placer dans la Liste de confiance.
Trouver le certificat de l'émetteur	Bad_CertificateUntrusted AuditCertificateUntrustedEventType	Un Certificat ne peut pas être fiable si le Certificat de l'émetteur est inconnu. Un Certificat auto-signé est son propre émetteur.
Signature	Bad_SecurityChecksFailed AuditCertificateInvalidEventType	Un Certificat avec une signature non valide doit toujours être rejeté.
Trouver la Liste d'annulation	Bad_CertificateRevocationUnknown Bad_CertificateIssuerRevocationUnknown AuditCertificateRevokedEventType	Chaque Certificat de CA peut avoir une liste d'annulation. Cette vérification échoue si cette liste n'est pas disponible (à savoir, une interruption du réseau empêche l'Application d'accéder à la liste). Aucune erreur n'est rapportée si l'Administrateur désactive les vérifications d'annulation pour un Certificat de CA. Cette erreur peut être supprimée.
Vérification d'annulation	Bad_CertificateRevoked Bad_CertificateIssuerRevoked AuditCertificateRevokedEventType	Le Certificat a été annulé et ne peut pas être utilisé. Cette erreur ne peut pas être supprimée.

Les Certificats sont en général placés dans un lieu central appelé CertificateStore. La Figure 20 illustre les interactions entre l'*Application*, l'*Administrateur* et le *CertificateStore*. Le *CertificateStore* peut être sur la machine locale ou dans quelque serveur central. Les mécanismes exacts utilisés pour accéder au *CertificateStore* dépendent de l'application et de l'environnement ICP installé par l'*Administrateur*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

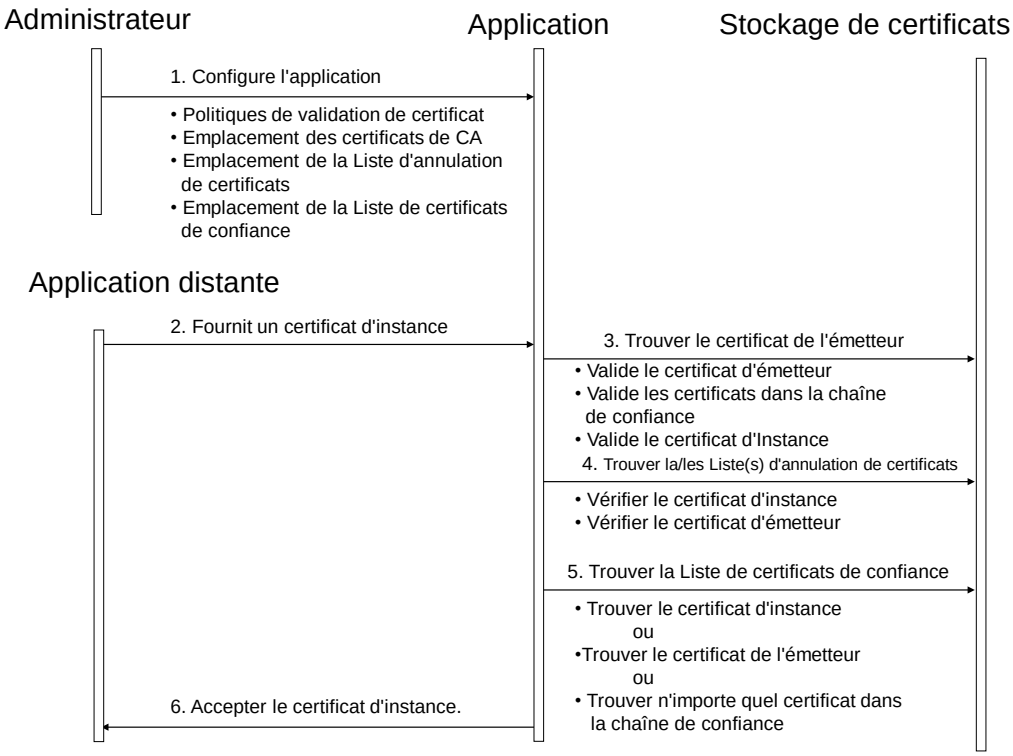


Figure 20 – Déterminer si un Certificat d'instance d'application est fiable

6.1.5 Valider un Certificat de logiciel

Les *Applications OPC UA* doivent valider les *SoftwareCertificates* fournies par les applications avec lesquelles elles communiquent.

Un *SoftwareCertificate* est valide si:

- la signature sur le *SoftwareCertificate* est valide;
- le *SoftwareCertificate* a dépassé sa date d'émission et il n'a pas expiré;
- le *SoftwareCertificate* n'a pas été annulé par l'émetteur;
- le *Certificate* d'émetteur est valide et n'a pas été annulé par la CA qui l'avait émis.

Les étapes utilisés pour valider des *SoftwareCertificates* sont pratiquement les mêmes que celles décrites pour les *Certificats* d'instance d'application en 6.1.4. La seule différence est que les *SoftwareCertificates* ne sont pas vérifiés par rapport à la *TrustList* de l'*Application*.

Les *SoftwareCertificates* contiennent toujours un *Certificat* de l'application qui est la propriété du vendeur qui distribue l'application. Le *Certificat* de l'application peut aussi être dans la chaîne des *Certificats* utilisée pour émettre des *Certificats* d'instance de l'application. Pour cette raison, les *Applications OPC UA* peuvent rejeter des *SoftwareCertificates* fournis par des applications si le *Certificat* de l'application ne fait pas partie de la chaîne de *Certificats* pour le *Certificat* d'instance de l'application. Il convient que les *Applications OPC UA* permettent aux *Administrateur s* d'exiger ce comportement. Les profils définis dans la CEI 62541-7 spécifient plus précisément la gestion attendue des *Certificats*.

6.1.6 Créer un Canal Sécurisé (SecureChannel)

Toutes les *Applications OPC UA* doivent établir un *Canal Sécurisé* avant de créer une *Session*. Ce *Canal Sécurisé* exige que les deux applications aient accès aux certificats qui

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

peuvent être utilisés pour chiffrer et signer l'échange des *Messages*. Les *certificats* d'instance de l'application installés en suivant le processus décrit en 6.1.2 peuvent être utilisés à cette fin.

Les étapes impliquées dans l'établissement d'un *Canal Sécurisé* sont montrées à la Figure 21.

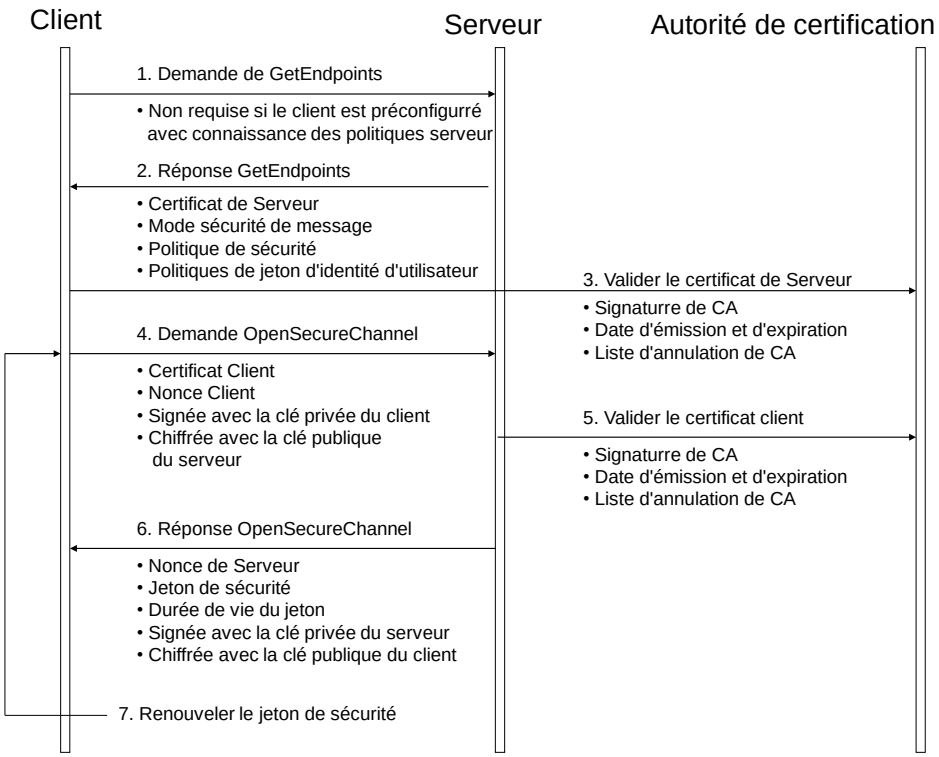


Figure 21 – Etablir un Canal Sécurisé

La Figure 21 ci-dessus suppose que le *Client* et le *Serveur* aient un accès en ligne à une autorité de certification (CA). Si un accès en ligne n'est pas disponible et si l'administrateur a installé une clé publique de CA sur la machine locale, le *Client* et *Serveur* doivent encore valider les certificats d'application en utilisant la clé en question. La figure montre une seule CA, mais il n'y a aucune obligation pour que les certificats du *Client* et du *Serveur* soient émis par la même autorité. Un certificat d'instance d'application auto-signé peut ne pas être vérifié par une CA. Tout certificat doit être rejeté s'il ne figure pas dans une liste de confiance fournie par l'administrateur.

Le *Client* et aussi le *Serveur* doivent avoir une liste de certificats à laquelle ils font confiance (parfois appelée Liste de certificats de confiance ou CTL). Ces certificats de confiance peuvent être des certificats pour des autorités de certification ou bien ils peuvent être des certificats d'instance d'*Applications OPC UA*. Les *Applications OPC UA* doivent être configurées pour refuser les connexions à des applications qui n'ont pas de certificats fiables.

L'intégrité des certificats peut être compromise, ce qui signifie qu'il convient de ne plus leur accorder de confiance. Les administrateurs peuvent annuler un certificat en le retirant de la liste de confiance pour toutes les applications ou bien la CA peut ajouter le certificat à la Liste d'annulation de certificats (CRL) pour le *Certificat* de l'émetteur. Les administrateurs peuvent sauvegarder une copie de la CRL pour chaque *Certificat* d'émetteur lorsque l'accès en ligne n'est pas disponible.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Un *Client* peut ne pas appeler *GetEndpoints* chaque fois qu'il se connecte au *Serveur*. Il convient que cette information change rarement et le *Client* peut la placer en cache localement. Si le *Serveur* rejette la demande *OpenSecureChannel*, il convient que le *Client* appelle *GetEndpoints* et s'assure que la configuration du *Serveur* n'a pas changé.

Il existe deux risques dont un *Client* doit être conscient lorsqu'il utilise le *Service GetEndpoints*. Le premier pourrait provenir d'un *Server Discovery* malveillant qui essaye de diriger le *Client* vers un *Serveur* malveillant. Pour cette raison, le *Client* doit vérifier que le *ServerCertificate* dans l'*EndpointDescription* est un *Certificate* fiable avant d'appeler *CreateSession*.

Le second risque à la sécurité provient d'une tierce partie qui altère le contenu des *EndpointDescriptions* à mesure qu'elles sont transférées sur le réseau pour revenir au *Client*. Le Client s'en protège en comparant la liste des *EndpointDescriptions* retourné par le *GetEndpoints Service* à la liste retournée dans la *response CreateSession*.

Les mécanismes exacts pour utiliser le jeton de sécurité pour signer et chiffrer les *Messages* échangés sur le *Canal Sécurisé* sont décrits dans la CEI 62541-6. Le processus de renouvellement des jetons est également décrit en détail dans la CEI 62541-6.

Dans de nombreux cas, les certificats utilisés pour établir le *Canal Sécurisé* seront les certificats d'instance d'application. Cependant, certaines *Piles de Communication* pourraient ne pas prendre en charge des certificats qui sont spécifiques à une seule application. Elles attendent plutôt que toute la communication soit sécurisée par un certificat spécifique à un utilisateur ou à toute la machine. Pour cette raison, les *Applications OPC UA* auront besoin d'échanger leurs certificats d'instance d'application pour créer une *Session*.

6.1.7 Créer une Session

Une fois qu'un *Client* OPC UA a établi un *Canal Sécurisé* avec un *Serveur*, elle peut créer une *Session* OPC UA.

Les étapes impliquées dans l'établissement d'une *Session* sont montrées à la Figure 22.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

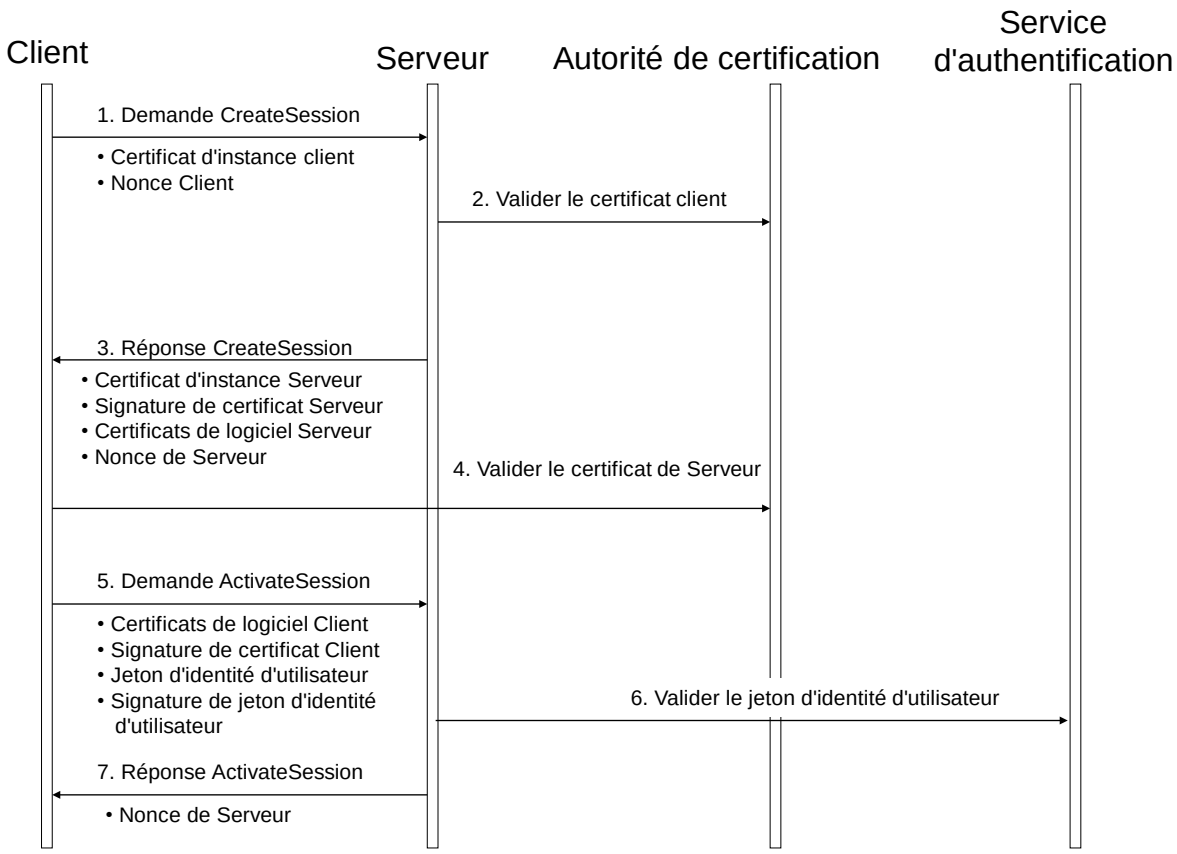


Figure 22 – Etablir une Session

La Figure 22 ci-dessus illustre les interactions entre un *Client*, un *Serveur*, une autorité de certification (CA) et un service d'authentification. La CA est chargé d'émettre les certificats d'application. Si le *Client* ou le *Serveur* n'a pas un accès en ligne à la CA, ils doivent valider les certificats d'application en utilisant la clé publique de la CA que l'administrateur doit installer sur la machine locale.

Le service d'authentification est une base de données centrale qui peut vérifier ce jeton d'utilisateur fourni par le *Client*. Ce service d'authentification peut également indiquer au *Serveur* les droits d'accès que l'utilisateur possède. Le service d'authentification dépend du jeton d'identité d'utilisateur. Il pourrait s'agir d'une autorité de certification, d'un service d'octroi de ticket Kerberos, d'un *Serveur* WS-Trust ou d'une base de données propriétaires quelconque.

Le *Client* et le *Serveur* doivent prouver la possession de leurs certificats d'application en signant les certificats avec un nonce accolé. Le mécanisme exact utilisé pour créer les signatures de preuve de possession est décrit en 5.6.2. De même, le *Client* doit prouver la possession de certains types de jetons d'identité d'utilisateur en créant des signatures avec le secret associé au jeton.

6.1.8 Se substituer à un utilisateur

Une fois qu'un *Client* OPC UA a établi une *Session* avec un *Serveur*, il peut changer l'identité d'utilisateur associée à la *Session* en appelant le service *ActivateSession*.

Les étapes impliquées dans la substitution à un utilisateur sont montrées à la Figure 23.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

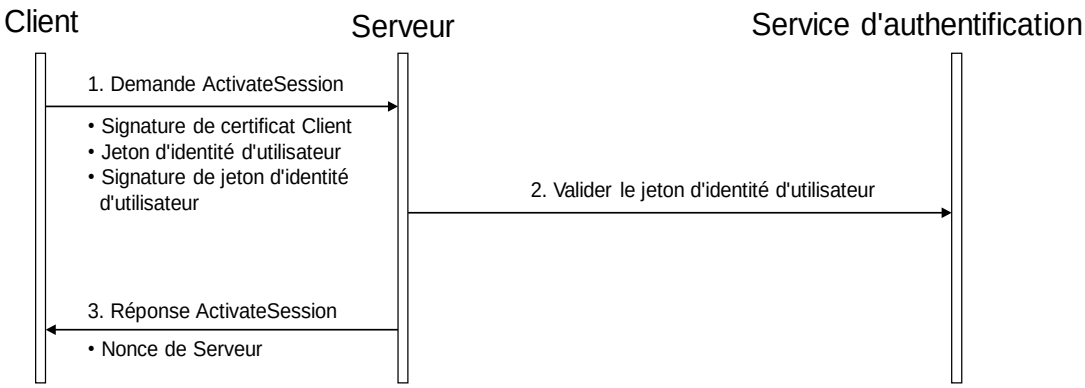


Figure 23 – Se substituer à un utilisateur

6.2 Audits

6.2.1 Vue générale

L'audit est une exigence dans de nombreux systèmes. Il fournit un moyen de suivre les activités qui ont lieu comme partie intégrante du fonctionnement normal du système. Il fournit aussi un moyen de suivre un comportement anormal. Il est également une exigence du point de vue de la sécurité. Pour plus d'informations sur les aspects sécuritaires des audits, voir la CEI/TR 62541-2. Ce paragraphe décrit ce qui est attendu d'un *Serveur* et d'un *Client* OPC UA quant aux audits et il détaille les exigences relatives à l'audit pour chaque jeu de services. L'audit peut être accompli en utilisant l'une des deux méthodes suivantes ou les deux:

a) L'*Application OPC UA* qui génère un événement d'audit peut journaliser l'entrée d'audit dans un fichier journal ou autre lieu de stockage.

b) Le *Serveur* OPC UA qui génère l'événement d'audit peut l'éditer en utilisant le mécanisme d'événements OPC UA. Cela permet à un *Client* OPC UA de s'abonner aux entrées d'audit et de les journaliser dans un fichier journal ou dans un autre lieu de stockage.

6.2.2 Journaux d'audits généraux

Chaque demande de *Service* OPC UA contient un paramètre chaîne qui est utilisé pour contenir un id d'enregistrement d'audit. Un *Client* ou tout *Serveur* opérant comme *Client*, tel qu'un *Serveur* combiné par exemple, peut créer une entrée locale de journal d'audits pour une demande qu'il présente. Ce paramètre permet à ce *Client* de transmettre l'identificateur pour cette entrée avec la demande. Si ce *Serveur* maintient aussi un journal d'audits, il convient qu'il inclue cet identificateur dans son entrée du journal d'audits qu'il écrit. Lorsque ce journal est examiné et l'entrée trouvée, l'examinateur pourra relier celle-ci directement à l'entrée de journal d'audits créée par le *Client*. Cette fonction permet la traçabilité à travers les journaux d'audits dans un système.

6.2.3 Événements d'audit généraux

Un *Serveur* qui maintient un journal d'audits doit fournir les entrées de journal d'audits via des *Messages* d'*Evénement*. L'*AuditEventType* et ses sous-types sont définis dans la CEI 62541-3. Un *Message* d'*Evénement* d'audit inclut aussi l'identificateur d'enregistrement d'audit. Les détails de l'*AuditEventType* et de ses sous-types sont définis dans la CEI 62541-5. Un *Serveur* qui est un *Serveur* combiné qui prend en charge les audits doit aussi s'abonner à des événements d'audit pour tous les *Serveurs* qu'il combine (en supposant qu'ils fournissent des audits). Il convient que le flux combiné soit disponible à partir du *Serveur* combiné.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.2.4 Audits pour jeu de services Discovery

Ce *Service Set* peut être divisé en deux groupes: les services appelés par les *Clients* OPC UA et les *Services* qui sont invoqués par des *Serveurs* OPC UA. Les *Services FindServers* et *GetEndpoints* qui sont appelés par les *Clients* OPC UA peuvent générer des entrées d'audit pour des invocations de *Service* non abouties. Le Service *RegisterServer* qui est invoqué par les *Serveurs* OPC UA doit générer des entrées d'audit pour tous les nouveaux enregistrements et pour les invocations de *Service* non abouties. Ces entrées d'audit doivent inclure l'URI de *Serveur* URI, les noms de *Serveur*, les URI *Discovery* et le statut isOnline. Il convient que des entrées d'audit ne soient pas générées pour l'invocation *RegisterServer* qui n'induit pas des changements aux *Serveurs* enregistrés.

6.2.5 Audits pour jeu de services SecureChannel

Tous les *Services* dans ce *Service Set* pour des *Servers* qui prennent en charge les audits peuvent générer des entrées d'audit et doivent générer des *Evénements* d'audit pour les invocations de service non abouties et pour l'invocation réussie des Services *OpenSecureChannel* et *CloseSecureChannel*. Il convient que les entrées d'audit générées par le *Client* soient installées avant l'appel effectif, permettant la fourniture de l'id correct d'enregistrement d'audit. Le Service *OpenSecureChannel* doit générer un *Evénement* d'audit du type *AuditOpenSecureChannelEventType* ou d'un sous-type de celui-ci. Le service *CloseSecureChannel* doit générer un *Evénement* d'audit du type *AuditCloseSecureChannelEventType* ou d'un sous-type de celui-ci. Tous ces deux types d'*Evénement* sont des sous-types de l'*AuditChannelEventType*. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et paramètres supplémentaires. Pour les cas d'échec, il convient d'inclure dans le *Message* pour les *Evénements* de ce type une description des raisons de l'échec du service. Il convient que la description soit plus détaillée que celle qui a été retourné au client. D'un point de vue sécurité, un client a seulement besoin de savoir qu'il y a eu échec. En revanche, du point de vue de l'*Audit*, il est nécessaire de connaître les détails exacts de l'échec. Dans le cas d'erreurs de validation de *Certificat*, il convient d'inclure dans la description l'*AuditEventId* de l'*AuditCertificateEvent* spécifique qui avait été généré pour rapporter l'erreur de certificat. L'*AuditCertifcateEvent* doit aussi contenir dans le détail l'erreur de validation de certificat. Il convient d'inclure dans les paramètres supplémentaires les détails de la demande. Il est bien entendu que ces événements peuvent être générés dans de nombreux cas par les *Piles de Communication* sous-jacentes, mais ils doivent être à la disposition du *Serveur* et le *Serveur* doit les consigner dans un rapport.

6.2.6 Audits pour jeu de services Session

Tous les *Services* de ce *Service Set* pour les *Serveurs* qui prennent en charge les audits peuvent générer des entrées d'audit et doivent générer des *Evénements* d'audit pour les invocations de *Service* abouties et non abouties. Ces *Services* doivent générer un *Evénement* d'audit du type *AuditSessionEventType* ou d'un sous-type de celui-ci. En particulier, ils doivent générer l'*EventType* de base ou le sous-type approprié, en fonction du service qui avait été invoqué. Le service *CreateSession* doit générer des événements de l'*AuditCreateSessionEventType* ou des sous-types de celui-ci. Le service *ActivateSession* doit générer des événements de l'*AuditActivateSessionType* ou des sous-types de celui-ci. Lorsque le *Service ActivateSession* est appelé pour changer l'identité d'utilisateur, le serveur doit générer des événements de l'*AuditImpersonateUserEventType* ou sous-types de celui-ci. Le service CloseSession service doit générer l'*EventType* de l'*AuditSessionEventType* ou sous-types de celui-ci. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et paramètres supplémentaires. Pour les cas d'échec, il convient d'inclure dans le *Message* pour les *Evénements* de ce type une description des raisons de l'échec du *Service*. Il convient d'inclure dans les paramètres supplémentaires les détails de la demande.

Ce *Service Set* doit également générer des événements d'audit supplémentaires lorsqu'il se produit des erreurs de validation de *Certificat*. Ces événements d'audit sont générés en plus des *AuditSessionEvents*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Pour les *Clients* prenant en charge les audits, l'accès aux services dans le *Session Service Set* doit générer des entrées d'audit pour les invocations abouties et non abouties du *Service*. Il convient que ces entrées d'audit soient installées avant l'invocation effective du *Service*, permettant à l'invocation de contenir l'id correct d'enregistrement d'audit.

6.2.7 Audits pour jeu de services NodeManagement

Tous les Services de ce *Service Set* pour les *Serveurs* qui prennent en charge les audits peuvent générer des entrées d'audit et doivent générer des *Evénements* d'audit pour les invocations de *Service* abouties et non abouties. Ces *Services* doivent générer un *Evénement* d'audit du type *AuditNodeManagementEventType* ou sous-types de celui-ci. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et paramètres supplémentaires. Pour les cas d'échecs, il convient d'inclure dans le *Message* pour les *Evénements* de ce type une description des raisons de l'échec du service. Il convient d'inclure dans les paramètres supplémentaires les détails de la demande.

Pour les *Clients* prenant en charge les audits, l'accès aux *Services* dans le *NodeManagement Service Set,* doit générer des entrées d'audit pour les invocations abouties et non abouties du *Service*. Il convient que toutes les entrées d'audit soient installées avant l'invocation effective du *Service*, permettant à l'invocation de contenir l'id correct d'enregistrement d'audit.

6.2.8 Audits pour jeu de services Attribute

Les Services *Write* ou *HistoryUpdate* de ce *Service Set* pour les *Serveurs* qui prennent en charge les audits peuvent générer des entrées d'audit et doivent générer des *Evénements* d'audit pour les invocations de *Service* abouties et non abouties. Ces *Services* doivent générer un *Evénement* d'audit du type *AuditUpdateEventType* ou sous-types de celui-ci. En particulier, le *Service Write* doit générer un événement d'audit du type *AuditWriteUpdateEventType* ou d'un sous-type de celui-ci. Le *Service HistoryUpdate* doit générer un *Event* d'audit du type *AuditHistoryUpdateEvent* ou d'un sous-type de celui-ci. Trois sous-types de l'*AuditHistoryUpdateEvent* sont définis, à savoir *AuditHistoryEventUpdateEventType*, *AuditHistoryValueUpdateEventType* et *AuditHistoryDeleteEventType*. Le sous-type dépend du type d'opération effectuée, à savoir mise à jour d'événements historiques, mise à jour de valeurs de données historiques ou suppression d'historique. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et des paramètres supplémentaires. Pour les cas d'échec, il convient d'inclure dans le *Message* pour les *Evénements* de ce type une description des raisons de l'échec du *Service*. Il convient d'inclure dans les paramètres supplémentaires les détails de la demande.

Les *Services Read* et *HistoryRead* peuvent générer des entrées d'audit et des *Evénements* d'audit pour les invocations de *Service* qui n'ont pas abouti. Il convient que ces *Services* génèrent un *Evénement* d'audit du type *AuditEventType* ou d'un sous-type de celui-ci. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et paramètres supplémentaires. Il convient d'inclure dans le *Message* pour les *Evénements* de ce type une description des raisons de l'échec du *Service*.

Pour les *Clients* prenant en charge les audits, l'accès aux services *Write* ou *HistoryUpdate* dans l'*Attribute Service Set* doit générer des entrées d'audit pour les invocations abouties et non abouties du *Service*. Les invocations des autres *Services* dans ce *Service Set* peuvent générer des entrées d'audit. Il convient que toutes les entrées d'audit soient installées avant l'invocation effective du *Service*, permettant à l'invocation de contenir l'id correct d'enregistrement d'audit.

6.2.9 Audits pour jeu de services Method

Tous les *Services* de ce *Service Set* pour les *Serveurs* qui prennent en charge les audits peuvent générer des entrées d'audit et doivent générer des *Evénements* d'audit pour les invocations abouties et non abouties si l'invocation modifie l'espace adresse, écrit une valeur ou modifie l'état du système (acquittement d'alarme, séquencement par lots ou autre

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

modifications de système). Ces appels de méthodes doivent générer un *Evénement* d'audit du type *AuditUpdateMethodEventType* ou sous-types de celui-ci. Les méthodes qui ne modifient pas *l'Espace Adresses*, n'écrivent pas de valeurs ou ne modifient pas l'état du système peuvent générer des événements. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et des paramètres supplémentaires.

Pour les *Clients* prenant en charge les audits, l'accès au *Method Service Set* doit générer des entrées d'audit pour les invocations abouties et non abouties du *Service* si l'invocation modifie l'espace adresse, écrit une valeur ou modifie l'état du système (acquittement d'alarme, séquencement par lots ou autre modifications de système). Les invocations des autres *Méthodes* peuvent générer des entrées d'audit. Il convient que toutes les entrées d'audit soient installées avant l'invocation effective du *Service*, permettant à l'invocation de contenir l'id correct d'enregistrement d'audit.

6.2.10 Audits pour le View Service Set, le Query Service Set, le MonitoredItem Service Set et le Subscription Service Set

Tous les *Services* de ces quatre *Service Sets* fournissent seulement des informations au *Client*, avec l'exception du *Service TransferSubscriptions* dans le *Subscription Service Set*. En général, ces services ne génèrent pas d'entrées d'audit ou de *Messages* d'événement d'audit. Le *Service TransferSubscriptions* doit générer un *Evénement* d'audit du type *AuditSessionEventType* ou sous-types de celui-ci pour les invocations de *Service* abouties et non abouties. Voir la CEI 62541-5 pour l'affectation détaillée des paramètres *SourceNode*, *SourceName* et des paramètres supplémentaires. Pour les cas d'échec, il convient d'inclure dans le *Message* pour les Événements de ce type une description des raisons de l'échec du service.

Pour les *Clients* prenant en charge les audits, l'accès au *Service TransferSubscriptions* dans le *Subscription Service Set* doit générer des entrées d'audit pour les invocations abouties et non abouties du *Service*. Les invocations des autres *Services* dans ce *Service Set* n'exigent pas d'entrées d'audit. Il convient que toutes les entrées d'audit soient installées avant l'invocation effective du *Service*, permettant à l'invocation de contenir l'id correct d'enregistrement d'audit.

6.3 Redondance

6.3.1 Vue d'ensemble de la redondance

La redondance dans OPC UA garantit que les Clients tout comme le *Serveur* peuvent être redondants. OPC UA ne fournit pas la redondance, elle fournit les structures de données et les services permettant d'obtenir la redondance d'une manière normalisée.

6.3.2 Vue d'ensemble de la redondance du serveur

6.3.2.1 Généralités

La redondance du serveur se présente sous deux modes, transparent et non transparent. Par définition, dans la redondance transparente, le basculement des responsabilités de *Serveur* d'un *Serveur* à un autre est transparent au *Client*: le *Client* ne se soucie pas de savoir, ou même ne sait pas, qu'un basculement a eu lieu; le *Client* n'a pas besoin de faire quoi que ce soit pour maintenir le flux de données. En revanche, le basculement non transparent exige une action du *Client*.

Les deux domaines où la redondance crée des besoins spécifiques sont le maintien de la synchronisation des informations du *Serveur* et du *Client* parmi les *Serveurs* et la régulation du basculement de flux de données d'un *Serveur* à l'autre.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.3.2.2 Redondance transparente

Pour la redondance transparente, tout ce qu'OPC UA fournit se résume aux structures de données pour permettre au *Client* d'identifier les *Serveurs* qui sont disponibles dans le jeu redondant, le niveau de service de chaque *Serveur* et le *Serveur* qui prend actuellement en charge une *Session* spécifiée. Toutes les interactions OPC UA au sein d'une session donnée doivent être prises en charge par un *Serveur* donné et le *Client* est à même d'identifier le *Serveur* en question, ce qui permet une piste d'audit complète pour les données. Il est de la responsabilité des *Serveurs* d'assurer que les informations sont synchronisées entre les *Serveurs* et qu'un autre *Serveur* reprendra la *Session* et les *Souscriptions* de l'adresse d'un *Serveur* donné à un autre *Client* en cas de basculement. Le basculement peut exiger une reconnexion à la couche transport du *Client* mais l'URL d'*Endpoint* du *Serveur* ne doit pas changer.

La Figure 24 montre une configuration type de la redondance transparente.

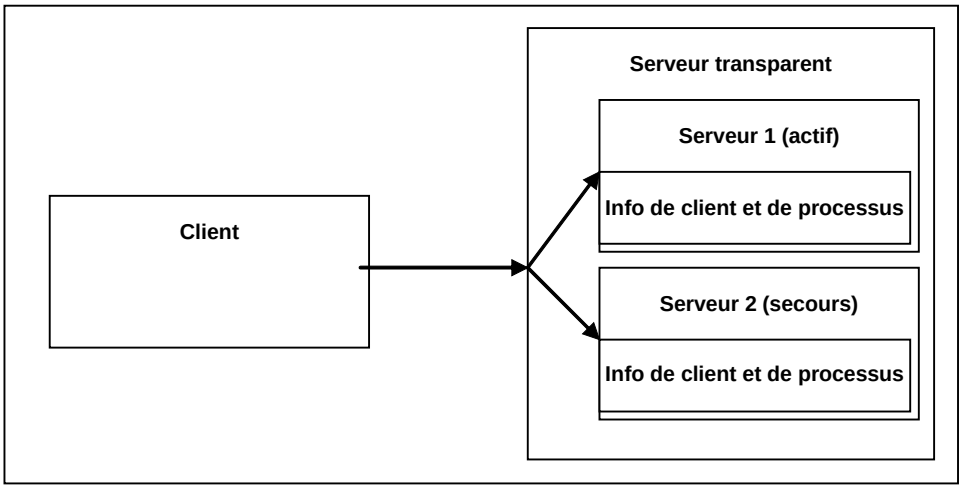


Figure 24 – Configuration type de la redondance transparente

6.3.2.3 Redondance non transparente

Pour la redondance non transparente, OPC UA fournit les mêmes structures de données et aussi les mêmes informations du *Serveur* qui précisent au *Client* les modes de basculement que le *Serveur* prend en charge. Ces informations permettent au *Client* de déterminer les actions qu'il peut avoir à prendre pour accomplir la reprise sur défaillance.

La Figure 25 montre une configuration type de la redondance non transparente.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

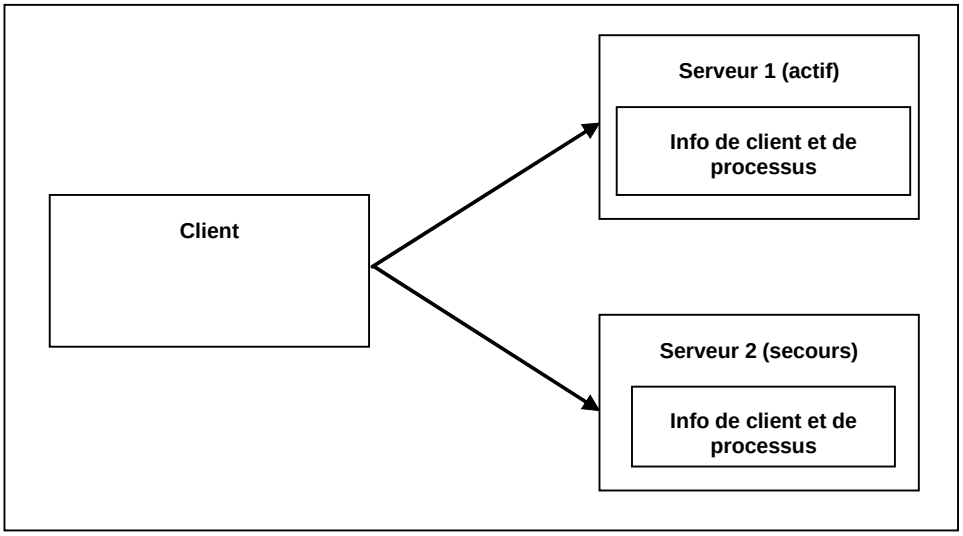


Figure 25 – Configuration type de la redondance non transparente

Pour la redondance non transparente, le *Serveur* a des concepts supplémentaires de basculement à froid, tiède et à chaud. Les basculements à froid concernent les *Serveurs* parmi lesquels un seul *Serveur* peut être actif à la fois. Les basculements tièdes concernent les *Serveurs* parmi lesquels les *Serveurs* de secours peuvent être actifs mais ne peuvent pas se connecter à des points de données réels (typiquement un système où les dispositifs sous-jacents sont limités à une seule connexion). Les basculements à chaud concernent les *Serveurs* parmi lesquels plus d'un *Serveur* peuvent être actifs et pleinement opérationnels.

Le Tableau 102 définit la liste des actions de basculement.

Tableau 102 – Actions de basculement de redondance

Mode de basculement	**COLD** (à froid)	**WARM** (tiède)	**HOT** (à chaud)
Sur connexion initiale:			
Connexion à plus d'un *Serveur* OPC UA		X	X
Création de *Souscriptions* et ajout d'éléments surveillés à ceux-ci		X	X
Activation de l'échantillonnage sur les *Souscriptions*			X
Au basculement:			
Connexion au *Serveur* OPC UA de secours	X		
Création de *Souscriptions* et ajout d'éléments surveillés	X		
Activation de l'échantillonnage sur les *Souscriptions*	X	X	
Activation de l'édition	X	X	X

La totalité ou une partie de cette activité peut être poussée dans un *Serveur* proxy sur la machine du *Client*, pour réduire la quantité de fonctionnalité qui doit être conçue dans le *Client* et pour permettre à des *Clients* plus simples de tirer parti de la redondance non transparente. En utilisant le *Service TransferSubscriptions*, qui permet au *Client* de demander le déplacement d'un jeu de *Souscriptions* d'une *Session* à une autre, le vendeur de *Serveur* peut effectivement rendre le basculement transparent d'une partie d'un proxy /stub qui vit sur le *Client*. Il existe deux façons de le faire, l'une nécessitant un code dans le *Serveur* pour le prendre en charge et l'autre le faisant en totalité à partir du processus proxy du *Client*

Lorsque c'est le proxy du *Client* qui est utilisé, le proxy duplique tout simplement les *Souscriptions* et les modifications des *Souscriptions*, en transmettant les appels aux deux *Serveurs*, mais en n'activant que l'édition ou l'échantillonnage sur un *Serveur*. Lorsque le proxy détecte une défaillance, il active l'édition et/ou l'échantillonnage sur le *Serveur* de secours, juste comme l'aurait fait le *Client* s'il s'agissait d'un *Client* conscient de la redondance.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'autre méthode exige aussi un stub *Client*, mais dans ce cas le stub () est un processus de poids beaucoup plus léger. Dans ce mode, c'est le *Serveur* qui duplique toutes les *Souscriptions* dans l'autre *Serveur*, mais le point d'extrémité *Client* pour ces *Souscriptions* est le *Serveur* actif. Lorsque le module de remplacement (Stub) détecte que le *Serveur* actif a failli, il émet un appel *TransferSubscriptions* au *Serveur* de secours, déplaçant les *Souscriptions* de la *Session* appartenant au *Serveur* défaillant vers sa propre *Session*, et activant l'édition.

La Figure 26 montre la différence entre la redondance de proxy *Serveur* et de proxy *Client*.

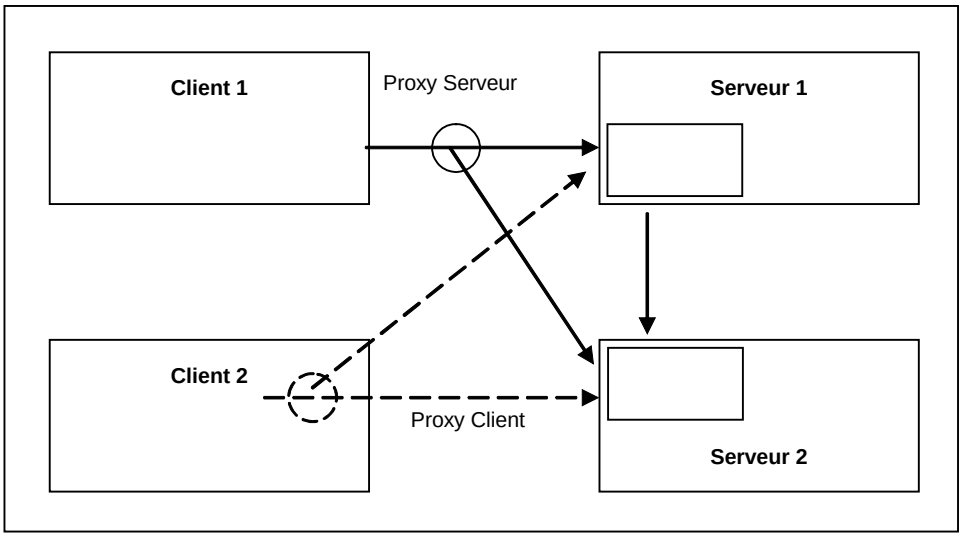


Figure 26 – Mode redondance

6.3.3 Redondance client

La redondance *Client* est prise en charge dans OPC UA par l'appel *TransferSubscriptions* et en expose les informations *Client* dans les structures d'informations *Serveur*. Sachant que la durée de vie d'une *Souscription* n'est pas liée à la *Session* dans laquelle il a été créé, les Clients de secours peuvent surveiller la *Session* du *Client* actif avec le *Serveur*, tout comme ils surveilleraient n'importe quelle autre variable de donnée. Si le *Client* actif cesse d'être actif, le *Serveur* doit envoyer une mise à jour de données à tout *Client* dont cette variable est surveillée. Lorsqu'il a reçu une telle notification, un *Client* de secours ordonne au *Serveur* de transférer les *Souscriptions* vers sa propre session. Si la *Souscription* est soigneusement conçue, avec des ressources suffisantes pour mémoriser les données lors de la permutation, il est nécessaire qu'il n'y ait pas de perte de données due à un basculement de *Client*.

OPC UA ne fournit pas de mécanisme normalisé pour acheminer le *SessionId* et les *SubscriptionIds* du *Client* actif vers les *Clients* secours, mais tant que les, *Clients* de secours connaissent le nom de *Client* du *Client* actif, ces informations sont immédiatement disponibles en utilisant les parties, *SessionDiagnostics* et *SubscriptionDiagnostics* des données de *ServerDiagnostics*.

7 Définitions de types de paramètres communs

7.1 Description d'une application (ApplicationDescription)

Les composantes de ce paramètre sont définies au Tableau 103.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 103 – ApplicationDescription

Nom	Type	Description
ApplicationDescription	structure	Spécifie une application qui est disponible.
applicationUri	String	L'identificateur unique au niveau global pour l'instance d'application
productUri	String	L'identificateur unique au niveau global pour le produit
applicationName	LocalizedText	Un nom descriptif localisé pour l'application.
applicationType	Enum ApplicationType	Le type d'application. Cette valeur est une énumération avec les valeurs suivantes: SERVER_0 L'application est un *Serveur*. CLIENT_1 L'application est un *Client*. CLIENTANDSERVER_2 L'application est un *Client* et un *Serveur*. DISCOVERYSERVER_3 L'application est un *DiscoveryServer*.
gatewayServerUri	String	Un URI qui identifie le *Gateway Server* associé aux *discoveryUrls*. Cette valeur n'est pas spécifiée si le *Serveur* n'est pas directement accessible. Ce champ n'est pas utilisé si l'*applicationType* est CLIENT_1.
discoveryProfileUri	String	Un URI qui identifie le profil de découverte pris en charge par les URL fournis. Ce champ n'est pas utilisé si l'*applicationType* est CLIENT_1. Si cette valeur n'est pas spécifiée, les points d'extrémité doivent prendre en charge les services Discovery Services définis en 5.4. Des profils de découverte de remplacement sont définis dans la CEI 62541-7.
discoveryUrls []	String	Une liste d'URL pour les *Endpoints* de découverte fournis par l'application. Si l'*applicationType* est CLIENT_1, ce champ doit contenir une liste vide.

7.2 Instanciation d'un certificat pour une application (ApplicationInstanceCertificate)

Un *ApplicationInstanceCertificate* est un *ByteString* contenant un *Certificat* codé. Le codage d'un *ApplicationInstanceCertificate* dépend de la mise en correspondance de la technologie de sécurité et il est complètement défini dans la CEI 62541-6. Le Tableau 104 spécifie les informations qui doivent être contenues dans un *ApplicationInstanceCertificate*.

Tableau 104 – ApplicationInstanceCertificate

Nom	Type	Description
ApplicationInstanceCertificate	structure	*ApplicationInstanceCertificate* avec une signature créée par une *Certificate Authority*.
version	String	Un identificateur pour la version du codage du *Certificat*.
serialNumber	ByteString	Un Identificateur unique pour le *Certificat* affecté par son Émetteur
signatureAlgorithm	String	L'algorithme utilisé pour signer le *Certificat*. La syntaxe de ce champ dépend du codage du *Certificat*.
signature	ByteString	La signature créée par l'Émetteur.
issuer	Structure	Un nom qui identifie le *Certificat* de l'Emetteur utilisé pour créer la signature.
validFrom	UTCTime	Le moment où le *Certificat* devient valide.
validTo	UTCTime	Le moment où le *Certificat* expire.
subject	Structure	Un nom qui identifie l'instance d'application que le *Certificat* décrit. Ce champ doit contenir le *productName* et le nom de l'organisation responsable de l'instance d'application.
applicationUri	String	L'*applicationUri* spécifié dans l'*ApplicationDescription*. L'*ApplicationDescription* est décrit en 7.1.
hostnames []	String	Le nom de la machine où l'instance d'application s'exécute. Une machine peut avoir plusieurs noms si elle est accessible via plusieurs réseaux. Le hostname (nom d'hôte) peut être une adresse réseau numérique ou un nom descriptif. Les *Certificats* du *Serveur* doivent avoir au moins un hostname défini.
publicKey	ByteString	La clé publique associée au *Certificat*.
keyUsage []	String	Spécifie comment la clé de *Certificat* peut être utilisée. Les *ApplicationInstanceCertificates* doivent prendre en charge Digital Signature, Non-Repudiation Key Encryption, Data Encryption et Client/Server Authorization. Le contenu de ce champ dépend du codage du *Certificat*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.3 Parcourir les résultats (BrowseResult)

Les composantes de ce paramètre sont définies au Tableau 105.

Tableau 105 – BrowseResult

Nom	Type	Description
BrowseResult	structure	Les résultats d'une opération Browse (navigation)
statusCode	StatusCode	Le statut de la *BrowseDescription.* Cette valeur est mise à *Good_MoreReferencesExist* s'il y a encore des références à retourner pour la *BrowseDescription*.
continuationPoint	ContinuationPoint	Une valeur opaque définie par le *Serveur* qui identifie le point de continuation. Le type *ContinuationPoint* est défini en 7.6.
References []	ReferenceDescription	Le jeu de références qui satisfait aux critères spécifiés dans la *BrowseDescription*. Vide, s'il n'y a pas de *Références* satisfaisant aux critères. Le type Reference Description est défini en 7.24.

7.4 Composantes d'un filtre (ContentFilter)

7.4.1 Composantes de la structure d'un filtre (ContentFilter structure)

La structure *ContentFilter* définit un ensemble d'éléments qui composent des critères de filtrage. Chaque élément dans l'ensemble décrit un opérateur et une matrice d'opérandes à utiliser par l'opérateur. Les opérateurs qui peuvent être utilisés dans un ContentFilter sont décrits au Tableau 110. Le filtre est contrôlé par l'évaluation de la première entrée dans la matrice d'éléments en commençant par le premier opérande de la matrice d'opérandes. Les opérandes d'un élément peuvent contenir des *Références* à des sous-éléments faisant que l'évaluation se poursuive aux éléments référencés de la matrice d'éléments. Un élément qui ne peut pas être rattaché en retour à l'élément de départ est ignoré. Pour tout opérateur, les opérandes supplémentaires doivent se traduire par une erreur. L'Annexe B fournit des exemples utilisant la structure *ContentFilter*.

Le Tableau 106 définit la structure ContentFilter.

Tableau 106 – Structure ContentFilter

Nom	Type	Description
ContentFilter	structure	
elements []	ContentFilterElement	Liste des opérateurs et de leurs opérandes qui composent les critères de filtrage. Le filtre est évalué en commençant par la première entrée de cette matrice.
filterOperator	enum FilterOperator	L'opérateur de filtre à évaluer. L'énumération *FilterOperator* est définie au Tableau 110.
filterOperands []	Extensible Parameter FilterOperand	Opérandes utilisés par l'opérateur sélectionné. Le nombre et l'utilisation dépendent des opérandes définis au Tableau 110. Cette matrice a besoin d'une entrée au moins. Ce type de paramètre extensible est le type de paramètre *FilterOperand* spécifié en 7.4.4. Il spécifie la liste des valeurs valides de *FilterOperand*.

7.4.2 Composantes des résultats d'un filtre (ContentFilterResult)

Les composantes de ce type de donnée sont définies au Tableau 107.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 107 – Structure ContentFilterResult

Nom	Type	Description
ContentFilterResult	structure	Une structure qui contient toutes les éventuelles erreurs associées au filtre.
elementResults	ContentFilter ElementResult	Une liste de résultats pour les éléments individuels du filtre. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments dans le paramètre *ContentFilter*.
statusCode	StatusCode	Le code de statut pour un élément seul.
operandStatusCodes []	StatusCode	Une liste de codes de statut pour les opérandes dans un élément. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des opérandes dans le *ContentFilterElement.* Cette liste est vide s'il ne s'est pas produit d'erreurs d'opérande.
operandDiagnosticInfos []	DiagnosticInfo	Une liste d'informations de diagnostic pour les opérandes dans un élément. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des opérandes dans le *ContentFilterElement.* Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement des opérandes.
elementDiagnosticInfos []	DiagnosticInfo	Une liste d'informations de diagnostic pour les éléments individuels du filtre. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments dans le paramètre de demande *filter*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement des éléments.

Le Tableau 108 définit les valeurs pour le paramètre *statusCode* qui sont spécifiques à cette structure. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 108 – Codes de résultats ContentFilterResult

Id symbolique	Description
Bad_FilterOperandCountMismatch	Le nombre d'opérandes fournis pour l'opérateur de filtre était plus faible que prévu pour l'opérande fourni.
Bad_FilterOperatorInvalid	Un opérateur non reconnu a été fourni dans un filtre.
Bad_FilterOperatorUnsupported	Un opérateur valide était fourni, mais le serveur ne prévoit pas de prise en prise pour l'opérateur du filtre.

Le Tableau 109 définit les valeurs pour le paramètre operand*StatusCode* qui sont spécifiques à cette structure. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 109 – Codes de résultats d'opérande ContentFilterResult

Id symbolique	Description
Bad_FilterOperandInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_FilterElementInvalid	L'élément référencé n'est pas un élément valide dans le filtre de contenu.
Bad_FilterLiteralInvalid	Le littéral référencé n'est pas un basedatatype valide.
Bad_AttribteIdInvalid	L'identificateur d'attribut n'est pas un identificateur d'attribut valide dans le système.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_NotTypeDefinition	Le Nodeid fourni n'était pas un nodeid de définition de type.

7.4.3 Operateurs d'un filtre (FilterOperator)

Le Tableau 110 définit les opérateurs de base qui peuvent être utilisés dans un ContentFilter. Voir le Tableau 111 pour une description d'opérateurs avancés. Voir 7.4.4 pour une définition des opérandes.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 110 – Définition de FilterOperator de base

Opérateur	Nombre d'opérandes	Description
Equals_0	2	TRUE si operand[0] est égal à operand[1]. Si les opérandes sont de types différents, le système doit exécuter toute conversion implicite en un type commun. Cet opérateur décide FALSE si aucune conversion implicite n'est disponible et les opérandes sont de types différents. Cet opérateur retourne FALSE si la conversion implicite échoue. Voir le débat relatif à la préséance des types de données au Tableau 114 pour plus d'informations sur la façon de convertir des opérandes de types différents.
IsNull_1	1	TRUE si operand[0] est une valeur null.
GreaterThan_2	2	TRUE si operand[0] est supérieur à operand[1]. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en une valeur ordonnée. [1]: Tout opérande qui se résout en une valeur ordonnée. Les mêmes règles de conversion définies pour *Equals* s'appliquent.
LessThan_3	2	TRUE si operand[0] est inférieur à operand[1]. Les mêmes règles de conversion et restrictions définies pour *GreaterThan* s'appliquent.
GreaterThanOrEqual_4	2	TRUE si operand[0] est supérieur ou égal à operand[1]. Les mêmes règles de conversion et restrictions définies pour *GreaterThan* s'appliquent.
LessThanOrEqual_5	2	TRUE si operand[0] est inférieur ou égal à operand[1]. Les mêmes règles de conversion et restrictions définies pour *GreaterThan* s'appliquent.
Like_6	2	TRUE si operand[0] correspond à un motif défini par operand[1]. Voir le Tableau 112 pour la définition de la syntaxe des motifs. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un String. [1]: Tout opérande qui se résout en un String. Cet opérateur décide FALSE si aucun opérande ne peut se résoudre en une chaîne.
Not_7	1	TRUE si operand[0] est FALSE. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un booléen. Si l'opérande ne peut pas se résoudre en un booléen, le résultat est NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.
Between_8	3	TRUE si operand[0] est supérieur ou égal à operand[1] et inférieur ou égal à operand[2]. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en une valeur ordonnée. [1]: Tout opérande qui se résout en une valeur ordonnée. [2]: Tout opérande qui se résout en une valeur ordonnée. Si les opérandes sont de types différents, le système doit exécuter toute conversion implicite pour faire correspondre tous les opérandes à un type commun. Si aucune conversion implicite n'est disponible et les opérandes sont de types différents, le résultat particulier est FALSE. Voir le débat relatif à la préséance des types de données au Tableau 114 pour plus d'informations sur la façon de convertir des opérandes de types différents.
InList_9	2..n	TRUE si operand[0] est égal à l'un ou plusieurs des opérandes restants. L'opérateur Equals est évalué pour l'operand[0] et chaque opérande restant de la liste. Si une évaluation Equals quelconque est TRUE, InList retourne TRUE.
And_10	2	TRUE si operand[0] et operand[1] sont TRUE. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un booléen. [1]: Tout opérande qui se résout en un booléen. Si un opérande quelconque ne peut pas se résoudre en un booléen, il est considéré NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.
Or_11	2	TRUE si operand[0] ou operand[1] est TRUE. Les restrictions suivantes s'appliquent aux opérandes: [0]:

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Opérateur	Nombre d'opérandes	Description
		Tout opérande qui se résout en un booléen. [1]: Tout opérande qui se résout en un booléen. Si un opérande quelconque ne peut pas se résoudre en un booléen, il est considéré NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.
Cast_12	2	Convertit operand[0] en une valeur ayant un type de donnée avec un NodeId identifié par operand[1]. Les restrictions suivantes s'appliquent aux opérandes: [0]: N'importe quel opérande. [1]: N'importe quel opérande qui se résout en un NodeId ou ExpandedNodeId lorsque le *Noeud* est le *NodeClass DataType*. S'il y a une erreur de conversion ou entachant l'un des paramètres, l'opération Cast évalue à NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.
BitwiseAnd_16	2	Le résultat est un nombre entier qui correspond à la taille du plus grand opérande et contient une opération And sur les bits appliquée aux deux opérandes lorsque tous deux ont été convertis à la même taille (celle du plus grand des deux opérandes) Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un nombre entier. [1]: Tout opérande qui se résout en un nombre entier. Si un opérande quelconque ne peut pas se résoudre en un nombre entier, il est considéré NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.
BitwiseOr_17	2	Le résultat est un nombre entier qui correspond à la taille du plus grand opérande et contient une opération Or sur les bits appliquée aux deux opérandes lorsque tous deux ont été convertis à la même taille (celle du plus grand des deux opérandes) Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un nombre entier. [1]: Tout opérande qui se résout en un Integer. Si un opérande quelconque ne peut pas se résoudre en un Integer, il est considéré NULL. Voir ci-dessous un débat relatif à la manipulation de NULL.

De nombreux opérandes ont des restrictions dues à leur type. Cela exige que l'opérande soit évalué pour déterminer son type. Dans certains cas, il s'agit du type spécifié dans l'opérande (à savoir un LiteralOperand). Dans d'autres cas, le type exige que la valeur d'un attribut soit lue. Les ElementOperands sont des valeurs booléennes, à moins que l'opérateur ne soit Cast ou un opérateur *RelatedTo* imbriqué.

Le Tableau 111 définit des opérateurs complexes qui exige un nœud cible (à savoir rangée) à évaluer. Ces opérateurs doivent être réévalués pour chaque nœud cible possible dans le jeu de résultats.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 111 – Définition de FilterOperator complexe

Opérateur	Nombre d'opérandes	Description
InView_13	1	TRUE si le *Noeud* cible est contenu dans la Vue définie par operand[0]. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un NodeId qui identifie une Vue du Nœud. Si operand[0] ne se résout pas en un NodeId qui identifie une Vue du Nœud , cette opération doit toujours être False.
OfType_14	1	TRUE si le *Noeud* cible est du type operand[0] ou d'un sous-type de operand[0]. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un NodeId qui identifie un Nœud ObjectType ou VariableType. Si operand[0] ne se résout pas en un NodeId qui identifie un Nœud ObjectType ou VariableType, cette opération doit toujours être False.
RelatedTo_15	6	TRUE si le *Noeud* cible est du type Operand[0] et est relié à un *NodeId* du type défini dans Operand[1] par le type *Référence* défini dans Operand[2]. Operand[0] ou Operand[1] peut aussi être un élément *Référence* où l'élément auquel il est fait référence est un autre opérateur RelatedTo. Cela permet le chaînage de relations (par exemple A est relié à B qui lui-même est relié à C). Dans ce cas, l'élément auquel il est fait référence retourne une liste de *NodeIds* au lieu de TRUE ou FALSE. Dans ce cas, si des erreurs apparaissent ou l'un quelconque des opérandes ne peut se résoudre en une valeur appropriée, le résultat de la relation chaînée est une liste vide de nœuds. Operand[3] définit le nombre de sauts qu'il convient que la relation suive. Si Operand[3] est 1, les objets doivent être reliés directement. Si un saut est supérieur à 1, un *NodeId* du type décrit dans l'Operand[1] est recherché à la profondeur spécifiée par le saut. Dans ce cas, le type de *Noeud* intermédiaire est indéfini, et seul le type *Référence* utilisé pour atteindre le *Noeud* final est défini. Si le nombre demandé de sauts ne peut pas être suivi, le résultat est FALSE, à savoir, une liste de Noeuds vide. Si Operand[3] est 0, la relation est suivie jusqu'à sa fin logique dans le sens direct et chaque *Noeud* est vérifié pour savoir s'il est du type spécifié dans Operand[1]. Si un *Noeud* quelconque satisfait à ces critères, le résultat est TRUE, à savoir, le *NodeId* est inclus dans une sous-liste. Operand[4] définit s'il convient que les opérandes [0] et [1] incluent la prise en charge des sous-types des types définis par ces opérandes. Un TRUE indique la prise en charge des sous-types. Operand[5] définit s'il convient qu'Operand[3] inclue la prise en charge pour les sous-types du type de référence défini par Operand[3]. Un TRUE indique la prise en charge des sous-types. Les restrictions suivantes s'appliquent aux opérandes: [0]: Tout opérande qui se résout en un NodeId ou ExpandedNodeId qui identifie un Nœud ObjectType ou VariableType ou une référence à un autre élément qui est un opérateur RelatedTo. [1]: Tout opérande qui se résout en un NodeId ou ExpandedNodeId qui identifie un Nœud ObjectType ou VariableType ou une référence à un autre élément qui est un opérateur RelatedTo. [2]: Tout opérande qui se résout en un NodeId qui identifie un Nœud ReferenceType. [3]: Tout opérande qui se résout en une valeur implicitement convertible en Int32. [4]: Tout opérande qui se résout en une valeur implicitement convertible en un booléen. Si l'opérande ne se résout pas en un booléen, une valeur FALSE est utilisée. [5]: Tout opérande qui se résout en une valeur implicitement convertible en un booléen. Si l'opérande ne se résout pas en un booléen, une valeur FALSE est utilisée. Si aucun des opérandes [0], [1], [2], [3] ne se résout en une valeur appropriée, le résultat de cette opération doit toujours être False (ou un jeu vide "Empty" dans le cas d'un opérande relatedTo imbriqué). Voir des exemples de RelatedTo en 7.4.4.

L'opérateur RelatedTo peut être utilisé pour identifier si un type donné, établi comme operand[1], est un sous-type d'un autre type établi comme operand[0] en mettant operand[2] au *ReferenceType HasSubtype* et operand[3] à 0.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'opérateur *Like* peut être utilisé pour effectuer des comparaisons de caractères de remplacement (Jokers). Plusieurs caractères spéciaux peuvent être inclus dans le second opérande de l'opérateur *Like*. Les caractères valides sont définis au Tableau 112. Les caractères de remplacement peuvent être combinés en une seule chaîne (à savoir 'Th[ia][ts]%' pourrait correspondre à 'That is fine', 'This is fine', 'That as one', 'This it is', 'Then at any', etc.).

Tableau 112 – Caractères de remplacement

Caractère spécial	Description
%	Correspond à toute chaîne de zéro ou plus de caractères (à savoir 'main%' correspondrait à toute chaîne qui commence par 'main', '%en%' correspondrait à toute chaîne qui contient les lettres 'en' telle que 'entail', 'green' et 'content'.) Si un signe '%' est prévu dans une chaîne, l'opérande de liste peut être utilisé (à savoir 5[%] correspondrait à '5%').
_	Correspond à tout caractère seul (à savoir '_ould' correspondrait à 'would', 'could'). Si un signe '_' ' est prévu dans une chaîne, l'opérande de liste peut être utilisé (à savoir 5[_] correspondrait à '5_').
\	Le caractère Escape (échappement) permet l'interprétation littérale (à savoir \\ est \, \% est %, _ est _).
[]	Correspond à tout caractère seul dans une liste (à savoir 'abc[13-68] correspondrait à 'abc1','abc3','abc4','abc5','abc6', et 'abc8'. 'xyz[c-f]' correspondrait à 'xyzc', 'xyzd', 'xyze', 'xyzf').
[^]	Ne correspondant à aucun caractère dans une liste. Le ^ doit être le premier caractère à l'intérieur du []. (à savoir 'ABC[^13-5]' NE correspondrait PAS à 'ABC1', 'ABC3', 'ABC4', et 'ABC5'. xyz[^dgh] NE correspondrait PAS 'xyzd', 'xyzg', 'xyzh'.)

Le Tableau 113 définit les règles de conversion pour les valeurs des opérandes. Les types sont automatiquement convertis si une conversion implicite existe (I). Si une conversion explicite existe (E), le type peut être converti avec l'opérateur cast. Si aucune conversion n'existe (X), les types ne peuvent pas être convertis, mais certains serveurs peuvent toutefois prendre en charge des conversions explicites spécifiques à l'application. Les types utilisés dans le tableau sont définis dans la CEI 62541-3. Un type de données qui ne figure pas dans le tableau n'a pas de conversions définies.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 113 – Règles de conversion

Type source	Type cible																				
	Boolean	Byte	ByteString	DateTime	Double	ExpandedNodeId	Float	Guid	Int16	Int32	Int64	NodeId	SByte	StatusCode	String	LocalizedText	QualifiedName	UInt16	UInt32	UInt64	XmlElement
Boolean	-	I	X	X	I	X	I	X	I	I	I	X	I	X	E	X	X	I	I	I	X
Byte	E	-	X	X	I	X	I	X	I	I	I	X	I	X	E	X	X	I	I	I	X
ByteString	X	X	-	X	X	X	X	E	X	X	X	X	X	X	X	X	X	X	X	X	X
DateTime	X	X	X	-	X	X	X	X	X	X	X	X	X	X	E	X	X	X	X	X	X
Double	E	E	X	X	-	X	E	X	E	E	E	X	E	X	E	X	X	E	E	E	X
ExpandedNodeId	X	X	X	X	X	-	X	X	X	X	X	E	X	X	I	X	X	X	X	X	X
Float	E	E	X	X	I	X	-	X	E	E	E	X	E	X	E	X	X	E	E	E	X
Guid	X	X	E	X	X	X	X	-	X	X	X	X	X	X	E	X	X	X	X	X	X
Int16	E	E	X	X	I	X	I	X	-	I	I	X	E	X	E	X	X	E	I	I	X
Int32	E	E	X	X	I	X	I	X	E	-	I	X	E	E	E	X	X	E	E	I	X
Int64	E	E	X	X	I	X	I	X	E	E	-	X	E	E	E	X	X	E	E	E	X
NodeId	X	X	X	X	X	I	X	X	X	X	X	-	X	X	I	X	X	X	X	X	X
SByte	E	E	X	X	I	X	I	X	I	I	I	X	-	X	E	X	X	I	I	I	X
StatusCode	X	X	X	X	X	X	X	X	X	I	I	X	X	-	X	X	X	E	I	I	X
String	I	I	X	E	I	E	I	I	I	I	I	E	I	X	-	E	E	I	I	I	X
LocalizedText	X	X	X	X	X	X	X	X	X	X	X	X	X	X	I	-	X	X	X	X	X
QualifiedName	X	X	X	X	X	X	X	X	X	X	X	X	X	X	I	I	-	X	X	X	X
UInt16	E	E	X	X	I	X	I	X	I	I	I	X	E	I	E	X	X	-	I	I	X
UInt32	E	E	X	X	I	X	I	X	E	I	I	X	E	E	E	X	X	E	-	I	X
UInt64	E	E	X	X	I	X	I	X	E	E	I	X	E	E	E	X	X	E	E	-	X
XmlElement	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	-

Les matrices d'un type source peuvent être converties en matrices d'un type cible par conversion de chaque élément. Une erreur de conversion pour un élément quelconque entraîne l'échec de toute la conversion.

Les matrices de longueur 1 peuvent être converties implicitement en une valeur scalaire du même type.

Guid, NodeIds et ExpandedNodeIds sont convertis en String, et vice versa, en utilisant la syntaxe définie dans la CEI 62541-6.

Les valeurs à virgule flottante sont arrondies en ajoutant 0,5 et en les tronquant lorsqu'elles sont converties en valeurs entières.

Les chaînes d'octets (ByteString) sont converties en chaînes (String) en formatant les octets sous la forme d'une séquence de chiffres hexadécimaux.

Les valeurs LocalizedText sont converties en String en abandonnant le Locale (paramètre de lieu). Les chaînes String sont converties en valeurs LocalizedText en mettant le Locale à “”.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les valeurs QualifiedName sont converties en String en abandonnant le NamespaceIndex (indice d'espace de noms). Les chaînes String sont converties en valeurs QualifiedName en mettant le NamespaceIndex à 0.

Un StatusCode peut être converti en UInt32 et Int32, et vice versa, en copiant les bits. Seuls les 16 bits de poids fort dans le StatusCode sont copiés lorsqu'il est converti en une valeur UInt16 ou Int16, et vice versa.

Les valeurs booléennes sont converties en '1' lorsqu'elles sont true (vraies) et en '0' lorsqu'elles sont false (fausses). Les valeurs numériques non nulles sont converties en valeurs booléennes vraies ("true"). Les valeurs numériques de 0 sont converties en valeurs booléennes false ("fausses"). Les valeurs String contenant "true", "false", "1" ou "0" peuvent être converties en valeurs booléennes. Les autres valeurs String (chaînes) provoquent une erreur de conversion. Dans ce cas, les chaînes sont sensibles à la casse.

Il est parfois possible d'utiliser des opérateurs cast implicites lorsque des opérandes ayant des types différents sont utilisés dans une opération. Dans cette situation, les règles de préséance définies au Tableau 114 sont utilisées pour déterminer la conversion implicite à utiliser. Le premier type de données dans la liste (de haut en bas) a la préséance la plus élevée. Si un type de données ne figure pas dans ce tableau, il ne peut pas être converti implicitement lors de l'évaluation d'une opération.

Par exemple, supposons que A = 1.1 (Float) et B = 1 (Int32) et que ces valeurs sont utilisées avec un opérateur Equals. Cette opération serait évaluée en forçant la conversion de la valeur Int32 en Float car le type de donnée Float a une préséance plus élevée.

Tableau 114 –Règles de préséance des données

Rang	Type de donnée
1	Double
2	Float
3	Int64
4	UInt64
5	Int32
6	UInt32
7	StatusCode
8	Int16
9	UInt16
10	SByte
11	Byte
12	Boolean
13	Guid
14	String
15	ExpandedNodeId
16	NodeId
17	LocalizedText
18	QualifiedName

Les opérandes peuvent contenir des valeurs null (à savoir des valeurs qui n'existent pas). Lorsque cela se produit, l'élément s'évalue toujours à NULL (sauf si l'opérateur IsNull a été spécifié). Le Tableau 115 définit comment combiner des éléments qui s'évaluent à NULL avec d'autres éléments dans une opération AND logique.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 115 – Table de vérité du AND (ET) logique

	TRUE (vrai)	**FALSE (faux)**	**NULL**
TRUE (vrai)	TRUE (vrai)	FALSE (faux)	NULL
FALSE (faux)	FALSE (faux)	FALSE (faux)	FALSE (faux)
NULL	NULL	FALSE (faux)	NULL

Le Tableau 116 définit comment combiner des éléments qui s'évaluent à NULL avec d'autres éléments dans une opération OR logique.

Tableau 116 – Table de vérité du OR (OU) logique

	TRUE (vrai)	**FALSE (faux)**	**NULL**
TRUE (vrai)	TRUE (vrai)	TRUE (vrai)	TRUE (vrai)
FALSE (faux)	TRUE (vrai)	FALSE (faux)	NULL
NULL	TRUE (vrai)	NULL	NULL

L'opérateur NOT évalue toujours à NULL s'il est appliqué à un opérande NULL.

Un ContentFilter qui évalue à NULL après que tous les éléments ont été évalués est évalué comme false.

7.4.4 Paramètre de FilterOperand

7.4.4.1 Vue générale

La structure *ContentFilter* spécifiée en 7.4 définit un ensemble d'éléments qui constituent les critères de filtrage et contient différents type de *FilterOperands*. Le paramètre *FilterOperand* est un paramètre extensible. Ce paramètre est défini au Tableau 117. Le type *ExtensibleParamter* est défini en 7.11.

Tableau 117 – TypeIds du paramètre FilterOperand

Id symbolique	**Description**
Element	Spécifie un indice dans la matrice d'éléments. Ce type est utilisé pour construire une arborescence logique de sous-éléments en reliant l'opérande d'un élément à un sous-élément.
Literal	Spécifie une valeur littérale.
Attribute	Spécifie n'importe quel *Attribut* d'un Noeud *Object* ou *Variable* utilisant un Noeud dans le système de type et le chemin relatif construits à partir de *ReferenceTypes* et de *BrowseNames*.
SimpleAttribute	Spécifie n'importe quel *Attribut* d'un Noeud *Object* ou *Variable* utilisant une *TypeDefinition* et un chemin relatif construit à partir de *BrowseNames*.

7.4.4.2 ElementOperand

L'*ElementOperand* fournit la liaison à des sous-éléments dans un *ContentFilter*. La liaison se présente sous la forme d'un entier qui est utilisé pour indexer dans la matrice d'éléments contenus dans le *ContentFilter*. Un indice est considéré valide si sa valeur est supérieure à l'indice d'élément dont if fait partie et il ne *Référence* pas un élément inexistant. Les *Clients* doivent construire des filtres de cette manière afin d'éviter des *Références* circulaires non valides. Il convient que les *Serveurs* se protègent des indices non valables en vérifiant l'indice avant de l'utiliser.

Le Tableau 118 définit le type *ElementOperand*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 118 – ElementOperand

Nom	Type	Description
ElementOperand	structure	Valeur d'ElementOperand.
index	UInt32	Indice dans la matrice d'éléments.

7.4.4.3 LiteralOperand

Le Tableau 119 définit le type *LiteralOperand*.

Tableau 119 – LiteralOperand

Nom	Type	Description
LiteralOperand	structure	Valeur de LiteralOperand.
value	BaseDataType	Une valeur littérale.

7.4.4.4 AttributeOperand

Le Tableau 120 définit le type *AttributeOperand*.

Tableau 120 – AttributeOperand

Nom	Type	Description
AttributeOperand	structure	Attribut d'un Noeud dans l'espace adresse.
nodeId	NodeId	*NodeId* d'un Noeud issu du système de type.
alias	String	Un paramètre facultatif utilisé pour identifier un pseudonyme ou s'y référer. Un pseudonyme est un nom symbolique qui peut être utilisé pour désigner cet opérande et l'utiliser dans un autre endroit de la structure de fichiers.
browsePath	RelativePath	Chemin de navigation relatif au Noeud identifié par le paramètre *nodeId*. Voir 7.25 pour la définition de *RelativePath*.
AttributeID	IntegerId	Identificateur de l'*Attribut*. Il doit être un identificateur d'*Attribut* valide. Le type *IntegerId* est défini en 7.13. Les IntegerIds pour les attributs sont définis dans la CEI 62541-6.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour une matrice. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini en 7.21. Ce paramètre n'est pas utilisé si l'Attribut spécifié n'est pas une matrice. Cependant, si l'Attribut spécifié est une matrice et ce paramètre n'est pas utilisé, tous les éléments doivent être inclus dans la plage. Ce paramètre est null s'il n'est pas utilisé.

7.4.4.5 SimpleAttributeOperand

Le *SimpleAttributeOperand* est une forme simplifiée de l'*AttributeOperand* et toutes les règles qui s'appliquent à l'*AttributeOperand* s'appliquent aussi au *SimpleAttributeOperand*. Les exemples fournis à l'Article B.1 utilisent seulement des *AttributeOperands*. Cependant, l'*AttributeOperand* peut être remplacé par un *SimpleAttributeOperand* chaque fois que tous les *ReferenceTypes* dans le *RelativePath* sont des sous-types des *HierarchialReferences* et les cibles sont des Nœuds *Object* ou *Variable et un pseudonyme (Alias) n'est pas requis.*

Le Tableau 121 définit le type *SimpleAttributeOperand*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 121 – SimpleAttributeOperand

Nom	Type	Description
SimpleAttributeOperand	structure	Attribut d'un Noeud dans l'espace adresse.
typeId	NodeId	*NodeId* d'un *TypeDefinitionNode*. Ce paramètre limite l'opérande à des instances du *TypeDefinitionNode* ou l'un de ses sous-types.
browsePath []	QualifiedName	Un chemin relatif vers un Noeud. Ce paramètre spécifie un chemin relatif utilisant une liste de *BrowseNames* en lieu et place de la structure *RelativePath* utilisée dans l'*AttributeOperand*. La liste des *BrowseNames* équivaut à un *RelativePath* qui spécifie des références directes qui sont des sous-types du *ReferenceType HierarchicalReferences*. Tous les *Noeuds* suivi par le *browsePath* doivent être de la *NodeClass Object* ou *Variable*. Si cette liste est vide, le *Noeud* est l'instance de la *TypeDefinition*.
attributeID	IntegerId	Identificateur de l'*Attribut*. L'*IntegerId* est défini en 7.13. L'*Attribute Value* doit être prise en charge par tous les *Serveurs*. La prise en charge d'autres *Attributs* dépend des exigences établies dans des Profils ou autres parties de cette spécification.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour une matrice. Le premier élément est identifié par l'indice 0 (zéro). Ce paramètre est ignoré si le Nœud sélectionné n'est pas une Variable ou la Valeur d'une Variable n'est pas une matrice. Ce paramètre est null s'il n'est pas spécifié. Toutes les valeurs dans la matrice sont utilisées si ce paramètre n'est pas spécifié. Le type *NumericRange* est défini en 7.21.

7.5 Compteur (Counter)

Ce type de donnée de primitive est un UInt32 qui représente la valeur d'un compteur. La valeur initiale d'un compteur est spécifiée par son utilisation. L'arithmétique modulaire est utilisée pour tous les calculs, le module y étant (max value + 1) Par conséquent,

$x + y = (x + y) \text{mod}(\text{max value} + 1)$

Par exemple:

max value + 1 = 0

max value + 2 = 1

7.6 Point de continuation (ContinuationPoint)

Un *ContinuationPoint* est utilisé pour suspendre une opération *Browse* ou *QueryFirst* et permette qu'elle soit relancée en appelant *BrowseNext* ou *QueryNext.* Les opérations sont suspendues lorsque le nombre de résultats trouvés dépasse la limite fixée par le *Client* ou le *Serveur*.

Le *Client* spécifie le nombre maximal de résultats par opération dans le message de demande. Un *Serveur* ne doit pas en retourner plus que ce nombre de résultats, mais il peut en retourner moins. Le *Serveur* attribue un *ContinuationPoint* s'il y a plus de résultats à retourner. Le *Serveur* doit toujours retourner au moins un résultat s'il retourne un *ContinuationPoint*.

Les *Serveurs* doivent prendre en charge au moins un *ContinuationPoint* par *Session*. Les *Serveurs* spécifient un nombre maximal de *ContinuationPoints* par *Session* dans l'*Object ServerCapabilities* défini dans la CEI 62541-5. Les *ContinuationPoints* restent actifs jusqu'à ce que le *Client* récupère les résultats restants, le *Client* libère le *ContinuationPoint* ou la *Session* est fermée. Le *Serveur* doit automatiquement libérer les *ContinuationPoints* des demandes précédentes s'ils sont nécessaires au traitement d'une demande. Le *Serveur*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

retourne une erreur *Bad_ContinuationPointInvalid* si un *Client* essaye d'utiliser un *ContinuationPoint* qui a été libéré.

Les demandes spécifient souvent plusieurs opérations qui peuvent ou peuvent ne pas nécessiter un *ContinuationPoint*. Un *Serveur* doit traiter les opérations jusqu'à ce qu'il n'ait plus de *ContinuationPoints*. Une fois que cela se produit, le *Serveur* doit retourner une erreur *Bad_NoContinuationPoints* pour les opérations restantes.

Un *Client* redémarre une opération en rendant le *ContinuationPoint* au *Serveur*. Il convient que le *Serveur* soit toujours capable de réutiliser le *ContinuationPoint* fourni et donc le *Serveurs* ne doit jamais retourner une erreur *Bad_NoContinuationPoints* lorsqu'il exécute une opération précédemment arrêtée.

Un *ContinuationPoint* est un sous-type du type de donnée *ByteString*.

7.7 Valeur d'une donnée (DataValue)

7.7.1 Généralités

Les composantes de ce paramètre sont définies au Tableau 122.

Tableau 122 – DataValue

Nom	Type	Description
DataValue	structure	La valeur et les informations associées.
value	BaseDataType	La valeur de donnée.
statusCode	StatusCode	Le *StatusCode* qui définit la capacité du *Serveur* à accéder/fournir la valeur. Le type *StatusCode* est défini en 7.33
sourceTimestamp	UTCTime	Marqueur temporel de la source pour la valeur.
sourcePicoSeconds	UInteger	Spécifie le nombre de 10 ps ($1{,}0 \times 10^{-11}$ s) qui doit être ajouté au sourceTimestamp.
serverTimestamp	UTCTime	Marqueur temporel du *Serveur* pour la valeur.
serverPicoSeconds	UInteger	Spécifie le nombre de 10 ps ($1{,}0 \times 10^{-11}$ s) qui doit être ajouté au serveurTimestamp.

7.7.2 PicoSeconds

Certaines applications requièrent des marqueurs temporels de haute résolution. Les champs *PicoSeconds* permettent aux applications de spécifier des marqueurs temporels avec une résolution de 10 ps. La taille réelle du champ *PicoSeconds* dépend de la résolution du *DataType UtcTime*. Par exemple, si le *DataType UtcTime* a une résolution de 100 ns, le champ *PicoSeconds* aurait à stocker des valeurs jusqu'à 10 000 afin de fournir la résolution de 10 ps. La résolution du *DataType UtcTime* (Type de donnée Temps TUC) dépend des *Mappings* (Mises en correspondance) définies dans la CEI 62541-6.

7.7.3 SourceTimestamp

Le *sourceTimestamp* est utilisé pour refléter le marqueur temporel qui avait été appliqué à la *Variable* value par la source de données. Une fois qu'une valeur a été affectée à un marqueur temporel de source, le marqueur temporel de source pour cette instance de valeur ne change jamais. Dans ce contexte, l'instance de valeur (Value instance) se réfère à la valeur reçue, quelle que soit sa valeur réelle.

Le *sourceTimestamp* doit être l'heure TUC et il convient qu'il indique l'heure du dernier changement de la *value* ou du *statusCode*.

Il convient que le *sourceTimestamp* soit généré le plus près possible de la source de la valeur, mais il est nécessaire que le marqueur temporel soit placé par la même horloge physique. Dans le cas des sources redondantes, il convient que les horloges des sources soient synchronisées.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Si le *Serveur* OPC UA reçoit la *Variable* value d'un autre *Serveur* OPC UA, le *Serveur* OPC UA doit toujours transmettre le marqueur temporel de la source sans changements. Si la source qui applique le marqueur temporel n'est pas disponible, le marqueur temporel de la source est mis à null. Par exemple, si une valeur ne pouvait pas être lue en raison d'une erreur au cours du traitement comme arguments invalides passés dans la demande, le sourceTimestamp doit être null.

Dans le cas d'un statut incertain ou mauvais, *sourceTimestamp* est utilisé pour refléter l'heure à laquelle la source a reconnu le statut autre que good (correct) ou l'heure de la dernière fois que le *Serveur* a essayé une reprise sur statut incertain ou mauvais.

Le *sourceTimestamp* est seulement retourné avec un *Attribute Value*. Pour tous les autres *Attributs*, le *sourceTimestamp* retourné est mis à null.

7.7.4 ServerTimestamp

Le *serveurTimestamp* est utilisé pour refléter l'heure à laquelle le *Serveur* a reçu une *Variable* value (Valeur) ou a su qu'elle était exacte.

Dans le cas d'un statut incertain ou mauvais, *serverTimestamp* est utilisé pour refléter l'heure à laquelle le *Serveur* a reçu le statut ou l'heure correspondant à la dernière fois que le *Serveur* a essayé une reprise sur statut incertain ou mauvais.

Dans le cas où le *Serveur* OPC UA s'abonne à une valeur issue d'un autre *Serveur* OPC UA, chaque *Serveur* applique son propre *serverTimestamp*. Ce cas de figure contraste avec celui du *sourceTimestamp* dans lequel seul l'émetteur des données est autorisé à appliquer le *sourceTimestamp*.

Si le *Serveur* s'abonne à la valeur issue d'un autre *Serveur* toutes les dix secondes et que la valeur change, le *serverTimestamp* est mis à jour chaque fois qu'une nouvelle valeur est reçue. Si la valeur ne change pas, les nouvelles valeurs ne seront pas reçues lors de la *Souscription*. Cependant, en l'absence d'erreurs, le *Serveur* de réception applique un nouveau *serverTimestamp* toutes les dix secondes car ne pas recevoir une valeur signifie que la valeur n'a pas changé. Ainsi, le *serverTimestamp* reflète l'heure à laquelle le *Serveur* a su que la valeur était exacte.

Ce concept s'applique aussi aux *Serveurs* OPC UA qui reçoivent des valeurs issues de sources de données pilotées par des exceptions. Par exemple, soient les hypothèses suivantes: le *Serveur* reçoit des valeurs issues d'un dispositif piloté par des exceptions et

a) le dispositif vérifie les valeurs toutes les 0,5 s,

b) la connexion au dispositif est bonne,

c) le dispositif a envoyé une mise à jour il y a 3 min avec une valeur de 1,234.

Dans ce cas, la valeur du *Serveur* serait de 1,234 et le *serverTimestamp* serait mis à jour toutes les 0,5 s après la réception de la valeur.

7.7.5 *StatusCode* affecté à une valeur

Le *StatusCode* est utilisé pour indiquer les conditions dans lesquelles une *Variable* value a été générée et, de ce fait, peut être utilisé comme un indicateur de validité de la valeur. Le *StatusCode* est défini en 7.33.

Condition globale (sévérité)

- Un *StatusCode* avec une sévérité Good signifie que la valeur est de bonne qualité.
- Un *StatusCode* avec une sévérité Uncertain signifie que la qualité de la valeur est incertaine pour des raisons indiquées par le Substatus (sous-statut).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- Un *StatusCode* avec une sévérité Bad signifie que la valeur n'est pas utilisable pour des raisons indiquées par le Substatus (sous-statut).

Règles

- Le *StatusCode* indique la validité d'une valeur. Par conséquent, il est requis que les *Clients* vérifient au minimum le *StatusCode Severity* de tous les résultats, même s'ils ne vérifient pas les autres champs, avant d'accéder à la valeur et de l'utiliser.
- Un *Serveur*, qui ne prend pas en charge les Informations sur le statut, doit retourner un code de sévérité Good. Il est également acceptable qu'un *Serveur* retourne simplement une sévérité et un sous-statut non spécifique (0).
- Si le *Serveur* n'a pas de valeur connue, notamment si la *Severity* est BAD - il doit retourner une valeur NULL.

7.8 Information de diagnostic (DiagnosticInfo)

Les composantes de ce paramètre sont définies au Tableau 123.

Tableau 123 – DiagnosticInfo

Nom	Type	Description
DiagnosticInfo	structure	Informations de diagnostic spécifiques au vendeur.
identifier	structure	L'identificateur spécifique au vendeur d'une erreur ou d'un état.
namespaceUri	Int32	L'identificateur symbolique défini par le paramètre *symbolicIdIndex* est défini dans le contexte d'un espace de noms. Cet espace de noms est représenté sous la forme d'une chaîne et est acheminé au *Client* dans le paramètre *stringTable* du paramètre *ResponseHeader* défini en 7.27. Le paramètre *namespaceIndex* contient l'indice dans la *stringTable* pour cette chaîne. −1 indique qu'aucune chaîne n'est spécifiée.
symbolicId	Int32	Une chaîne d'identificateur symbolique spécifique au vendeur identifie une erreur ou un état. La longueur maximale de cette chaîne est de 32 caractères. Il convient que les *Serveurs* souhaitant retourner un code de retour numérique convertissent le code de retour en chaîne et retournent la chaîne dans cet identificateur. La chaîne d'identificateur symbolique est acheminée au *Client* dans le paramètre *stringTable* du paramètre *ResponseHeader* défini en 7.27. Le paramètre *symbolicIdIndex* contient l'indice dans la *stringTable* pour cette chaîne. −1 indique qu'aucune chaîne n'est spécifiée.
locale	Int32	La partie paramètre de lieu du texte localisé spécifique au vendeur décrivant l'identificateur symbolique. La chaîne de texte localisé est acheminée au *Client* dans le paramètre *stringTable* du paramètre *ResponseHeader* défini en 7.27. Le paramètre *localizedTextIndex* contient l'indice dans la *stringTable* pour cette chaîne. −1 indique qu'aucune chaîne n'est spécifiée.
localizedText	Int32	Une chaîne de texte localisé spécifique au vendeur décrit l'identificateur symbolique. La longueur maximale de cette chaîne de texte est de 256 caractères. La chaîne de texte localisé est acheminée au *Client* dans le paramètre *stringTable* du paramètre *ResponseHeader* défini en 7.27. Le paramètre *localizedTextIndex* contient l'indice dans la *stringTable* pour cette chaîne. −1 indique qu'aucune chaîne n'est spécifiée.
additionalInfo	String	Informations de diagnostic spécifiques au vendeur.
innerStatusCode	StatusCode	Le *StatusCode* issu de l'opération interne. De nombreuses applications effectueront des appels dans des systèmes sous-jacents au cours du traitement de la demande OPC UA. Un *Serveur* OPC UA a l'option de consigner le statut issu du système sous-jacent dans les informations de diagnostic.
innerDiagnosticInfo	DiagnosticInfo	Les informations de diagnostic associées au *StatusCode* interne.

7.9 Description d'un point d'extrémité (EndpointDescription)

Les composantes de ce paramètre sont définies au Tableau 124.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 124 – EndpointDescription

Nom	Type	Description
EndpointDescription	structure	Décrit un *Endpoint* pour un *Serveur*.
endpointUrl	String	L'URL de l'*Endpoint* décrit.
server	ApplicationDescription	La description pour le *Serveur* auquel l'*Endpoint* appartient. Le type *ApplicationDescription* est défini en 7.1.
serverCertificate	ApplicationInstance Certificate	Le *Certificate* d'instance d'application émis au *Serveur*. Le type *ApplicationInstanceCertificate* est défini en 7.2.
securityMode	Enum MessageSecurityMode	Le type de sécurité à appliquer aux messages. Le type *MessageSecurityMode* est défini en 7.14. Un *Canal Sécurisé* peut devoir être créée même si le *securityMode* est NONE. Le comportement exact dépendant de la mise en correspondance utilisée et est décrit dans la CEI 62541-6.
securityPolicyUri	String	L'URI pour la *SecurityPolicy* à utiliser pour sécuriser des messages. Le jeu d'URI connus et les *SecurityPolicies* qui leur sont associées sont définis dans la CEI 62541-7.
userIdentityTokens []	UserTokenPolicy	Les jetons d'identité d'utilisateur que le *Serveur* acceptera. Le *Client* doit passer l'un des *UserIdentityTokens* dans la demande *ActivateSession*. Le type *UserTokenPolicy* est décrit en 7.36
transportProfileUri	String	L'URI du *Profile Transport* (profil de transport) pris en charge par l'*Endpoint*. La CEI 62541-7 définit les URI pour les *Transport Profiles* (profils de transport).
securityLevel	Byte	Une valeur numérique qui indique le niveau de sécurité de l'EndpointDescription en comparaison à d'autres EndpointDescriptions pour le même *Serveur*. Une valeur de 0 indique que l'EndpointDescription n'est pas recommandée et n'est prise en charge que pour la rétrocompatibilité.

7.10 Identifiant étendu d'un nœud (ExpandedNodeId)

Les composantes de ce paramètre sont définies au Tableau 125. Les *ExpandedNodeIds* permettent de spécifier explicitement l'espace de noms sous la forme d'une chaîne ou avec un indice dans le tableau d'espaces de noms du *Serveur*.

Tableau 125 – ExpandedNodeId

Nom	Type	Description
ExpandedNodeId	structure	Le *NodeId* avec l'espace de noms étendu à sa représentation sous forme de chaîne.
serverIndex	Index	Indice qui identifie le *Serveur* qui contient le *TargetNode*. Ce *Serveur* peut être le *Serveur* local ou un *Serveur* distant. Cet indice est l'indice du *Serveur* en question dans le tableau de *Serveurs* du *Serveur* local. L'indice du *Serveur* local dans le tableau de *Serveurs* est toujours 0. Tous les *Serveurs* distants ont des indices supérieurs à 0. Le tableau de *Serveurs* est contenu dans l'*Object Server* (objet serveur) dans l'Espace d'Adresses (voir la CEI 62541-3 et la CEI 62541-5). Le *Client* peut lire la *Serveur* table *Variable* (variable tableau de serveurs) pour accéder à la description du *Serveur* cible.
namespaceUri	String	L'URI de l'espace de noms. Si ce paramètre est spécifié, l'indice d'espace de noms est ignoré. Le paragraphe 5.4 et la CEI 62541-12 décrivent le mécanisme de découverte qui peut être utilisé pour résoudre les URI en des URL.
namespaceIndex	Index	L'indice dans le tableau d'espace de noms du *Serveur* Ce paramètre doit être 0 et est ignoré dans le *Serveur* si l'URI d'espace de noms est spécifié.
identifierType	IdType	Type de l'élément identificateur du *NodeId*.
identifier	*	L'identificateur pour un Noeud dans l'Espace d'Adresses d'un *Serveur* OPC UA (voir la définition de *NodeId* dans la CEI 62541-3).

7.11 Parametre extensible (ExtensibleParameter)

Les types de paramètres extensibles peuvent seulement être étendus par des parties supplémentaires de la série de normes.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'*ExtensibleParameter* définit une structure de données à deux éléments. Le *parameterTypeId* spécifie le codage de type de donnée du second élément. Par conséquent, le second élément est spécifié comme étant "--". Le type de base *ExtensibleParameter* est défini au Tableau 126.

Les paramètres extensibles concrets qui sont communs à OPC UA sont définis à l'Article 7. Des parties supplémentaires de la série de normes peuvent définir de types de paramètres extensibles supplémentaires.

Tableau 126 – Type de base ExtensibleParameter

Nom	Type	Description
ExtensibleParameter	structure	Spécifie les détails d'un type de paramètre extensible.
parameterTypeId	NodeId	Identifie le type de données du paramètre qui suit.
parameterData	--	Les détails pour le type de paramètre extensible.

7.12 Indexe (Index)

Ce type de données de primitive est un UInt32 qui identifie un élément d'une matrice.

7.13 IntegerId

Ce type de données de primitive est un UInt32 qui est utilisé comme identificateur, tel qu'un descripteur. Toutes les valeurs, autres que 0, sont valides.

7.14 Mode de sécurité des messages (MessageSecurityMode)

Le *MessageSecurityMode* est une énumération qui spécifie la sécurité qu'il convient d'appliquer aux échanges de messages au cours d'une Session. Les valeurs possibles sont décrites au Tableau 127.

Tableau 127 – Valeurs de MessageSecurityMode

Valeur	Description
INVALID_0	Le MessageSecurityMode n'est pas valide. Cette valeur est la valeur par défaut afin d'éviter qu'un choix accidentel de "no security" (aucune sécurité) ne soit appliqué. Ce choix sera toujours rejeté.
SIGN_1	Tous les messages sont signés, mais ne sont pas chiffrés.
SIGNANDENCRYPT_2	Tous les messages sont signés et chiffrés.

7.15 Surveillance des paramètres (MonitoringParameters)

Les composantes de ce paramètre sont définies au Tableau 128.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 128 – MonitoringParameters

Nom	Type	Description
MonitoringParameters	structure	Paramètres qui définissent les caractéristiques de surveillance relatives à un *MonitoredItem*.
clientHandle	IntegerId	Identificateur fourni par le *Client* pour le *MonitoredItem*. Cet identificateur est utilisé dans les *Notifications* générées pour le *Noeud* de liste. Le type *IntegerId* est défini en 7.13.
samplingInterval	Duration	L'intervalle qui définit la fréquence la plus rapide à laquelle il convient d'accéder et évaluer le(s) *MonitoredItem*(s). Cet intervalle est défini en millisecondes. La valeur 0 indique qu'il convient que le *Serveur* utilise la fréquence pratique la plus rapide. La valeur -1 indique que l'intervalle d'échantillonnage par défaut défini par la fréquence d'édition d'une *Souscription* est utilisé. Le *Serveur* utilise ce paramètre pour affecter les *MonitoredItems* à un intervalle d'échantillonnage qu'il prend en charge.
filter	Extensible Parameter MonitoringFilter	Un filtre utilisé par le *Serveur* pour déterminer s'il convient que le *MonitoredItem* génère une *Notification*. S'il n'est pas utilisé, ce paramètre est null. Le type de paramètre *MonitoringFilter* est un type de paramètre extensible spécifié en 7.16. Il spécifie les types de filtres qui peuvent être utilisés.
queueSize	Counter	La taille demandée de la file d'attente des *MonitoredItems*. Si des *Evénements* sont perdus, un *Evénement* du type *EventQueueOverflow* est généré. Les valeurs suivantes ont une signification spéciale: Valeur — Signification 1 — la file d'attente a une seule entrée, désactivant effectivement la mise en file d'attente. >1 — une file d'attente en "premier entré premier sorti" est à utiliser. Valeur maximum — la taille maximale que le *Serveur* peut prendre en charge. Elle est utilisée pour les *Notifications* d'*Evénements*. Dans ce cas, le *Serveur* a la responsabilité du tampon d'*Evénements*. Si 0 est passé par le client, le serveur retourne la taille de file d'attente par défaut à 1. Et le retour doit être 1 comme *revisedQueueSize* pour les éléments surveillés de données. Pour les éléments surveillés d'événements, la valeur est ignorée et le serveur doit établir la taille de file d'attente maximale prise en charge comme étant *revisedQueueSize*. Si des *Evénements* sont perdus, un *Evénement* du type *EventQueueOverflowEventType* est généré.
discardOldest	Boolean	Un paramètre booléen qui spécifie la politique de rejet lorsque la file d'attente est pleine et une nouvelle *Notification* est à placer en file d'attente. Ce paramètre a les valeurs suivantes: TRUE — La plus ancienne (première) *Notification* dans la file d'attente est rejetée. La nouvelle *Notification* est ajoutée à la fin de la file d'attente. FALSE — la nouvelle *Notification* est rejetée. La file d'attente est inchangée.

7.16 Paramètres des filtres de surveillance (MonitoringFilter)

7.16.1 Vue générale

Le *Service CreateMonitoredItem* permet de spécifier un filtre pour chaque *MonitoredItem*. Le *MonitoringFilter* est un paramètre extensible dont la structure dépend du type d'élément surveillé. Les *parameterTypeIds* sont déjà définis au Tableau 129. D'autres types peuvent être définis par des parties supplémentaires de la présente spécification à plusieurs parties ou par d'autres spécifications basées sur OPC UA. Le type *ExtensibleParamter* est défini en 7.11.

Chaque *MonitoringFilter* peut avoir une structure associée *MonitoringFilterResult* qui retourne des paramètres révisés et/ou des informations d'erreur aux clients dans la réponse. Les structures de résultats, lorsqu'elles existent, sont décrites dans la section qui définit le *MonitoringFilter*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 129 – TypeIds du paramètre MonitoringFilter

Id symbolique	Description
DataChangeFilter	Le changement apporté à une valeur de donnée qui doit entraîner la création d'une *Notification*.
EventFilter	Si une *Notification* est conforme au *EventFilter*, la *Notification* est envoyée au *Client*.
AggregateFilter	L'agrégat et ses intervalles lorsqu'il sera calculé et une Notification est générée.

7.16.2 DataChangeFilter

Le *DataChangeFilter* définit les conditions dans lesquelles il convient de rapporter une notification de changement de donnée et, facultativement, une plage ou une bande de valeurs lorsqu'il n'est pas généré de *Notification* de *DataChange*. Cette plage est appelée *Deadband* (bande morte). Le *DataChangeFilter* est défini au Tableau 130.

Tableau 130 – DataChangeFilter

Nom	Type	Description
DataChangeFilter	structure	
trigger	enum DataChangeTrigger	Spécifie les conditions dans lesquelles il convient de rapporter une notification de changement de donnée. Cette énumération a les valeurs suivantes: STATUS_0 Rapporter une notification SEULEMENT si le *StatusCode* associé à la valeur change. Voir le Tableau 166 pour les *StatusCodes* définis dans la présente norme. La CEI 62541-8 spécifie des *StatusCodes* supplémentaires qui sont en particulier valides pour des données de dispositif. STATUS_VALUE_1 Rapporter une notification si le *StatusCode* ou la valeur change. Le filtre *Deadband* peut être ajouté en plus pour filtrer les changements de valeur. **Il s'agit du réglage par défaut si aucun filtre n'est installé.** STATUS_VALUE_TIMESTAMP_2 Rapporter une notification si le StatusCode, la valeur ou le *SourceTimestamp* change. Le filtre *Deadband* peut être ajouté en plus pour filtrer les changements de valeur. Le DataChangeFilter n'est pas appliqué à l'élément surveillé, STATUS_VALUE_1 est le comportement par défaut pour la production de rapports.
deadbandType	UInt32	Une valeur qui définit le type et le comportement *Deadband*. Value deadbandType None_0 Il convient de n'appliquer aucun calcul *Deadband*. Absolute_1 AbsoluteDeadband (Voir ci-dessous) Percent_2 PercentDeadband (Ce type est spécifié dans la CEI 62541-8).
deadbandValue	Double	Le *Deadband* est appliqué uniquement si * le *trigger* inclut des changements de valeur et * le *deadbandType* est établi de manière appropriée. La bande morte est ignorée si le statut de l'élément de donnée change. ***DeadbandType = AbsoluteDeadband***: Pour ce type, la *deadbandValue* contient le changement absolu apporté à une valeur de donnée qui doit provoquer la création d'une *Notification*. Ce paramètre s'applique uniquement aux *Variable*s de n'importe quel type de donnée numérique. Une exception qui provoque une *Notification* de *DataChange* basée sur une AbsoluteDeadband est déterminée comme suit: Exception **si (valeur absolue de (dernière valeur en cache - valeur courante) > *AbsoluteDeadband*)** La dernière valeur en cache est définie comme étant la plus récente valeur envoyée à la voie de *Notifications*. Si l'élément est une matrice de valeurs, toute la matrice est retournée si un élément de matrice dépasse l'AbsoluteDeadband. ***DeadbandType = PercentDeadband:*** Ce type est spécifié dans la CEI 62541-8

Le *DataChangeFilter* n'a pas de structure de résultats associée.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.16.3 EventFilter

L'*EventFilter* permet le filtrage et la sélection de contenu des *Event Subscriptions*.

Si une *Notification* d'*Evénement* est conforme au filtre défini par le paramètre *where* de l'*EventFilter*, la *Notification* est envoyée au *Client*.

Chaque *Notification* d'*Event* doit inclure les champs définis par le paramètre *selectClauses* de l'*EventFilter*. Les *EventType*s définis sont spécifiés dans la CEI 62541-5.

Les paramètres *selectClause* et *whereClause* sont spécifiés avec la structure *SimpleAttributeOperand* (voir 7.4.4.5). Cette structure exige le *NodeId* d'un *EventType* pris en charge par le *Serveur* et un chemin vers une *InstanceDeclaration*. Une *InstanceDeclaration* est un Noeud qui peut être trouvé en suivant des références hiérarchiques directes à partir de l'*EventType* totalement hérité où le *Noeud* est aussi la source d'une référence *HasModellingRule*. Les *EventTypes*, les *InstanceDeclarations* et les *Modelling Rules* sont décrits de manière complète dans la CEI 62541-3.

Dans certains cas, le même *BrowsePath* s'appliquera à plusieurs *EventTypes*. Si le *Client* spécifie le *BaseEventType* dans le SimpleAttributeOperand, le *Serveur* doit évaluer le *BrowsePath* sans tenir compte du *Type*.

Chaque *InstanceDeclaration* dans le chemin doit être un *Node Object* ou *Variable*. Le *Noeud* final dans le chemin peut être un *Object Node*; cependant, les *Object Nodes* sont disponibles uniquement pour les *Evénements* qui sont visibles dans l'Espace d'Adresses du *Serveur*.

La structure *SimpleAttributeOperand* permet au Client de spécifier n'importe quel *Attribut*; cependant, le *Serveur* est seulement tenu de prendre en charge le *Value Attribute* pour des *Variable Nodes* et le *NodeId Attribute* pour des *Object Nodes*. Cela dit, les profils définis dans la CEI 62541-7 peuvent rendre obligatoire la prise en charge ud' *Attributs* supplémentaires.

La structure *SimpleAttributeOperand* est utilisée dans la *selectClause* pour sélectionner la valeur à retourner si un *Evénement* satisfait aux critères spécifiés par la *whereClause*. Une valeur null est retournée dans le champ événement correspondant dans la réponse Publish si le *field* sélectionné ne fait pas partie de l'*Evénement* ou une erreur a été retournée dans les *selectClauseResults* de l'*EventFilterResult*. Si le *field* sélectionné est disponible mais ne peut pas être retourné au *Client*, le *Serveur* doit retourner un *StatusCode* qui indique la raison de l'erreur. Par exemple, le *Serveur* doit retourner une erreur *Bad_UserAccessDenied* si la valeur n'est pas accessible à l'utilisateur associé à la *Session*. Si un *Value Attribute* a un *StatusCode* incertain ou mauvais qui lui est associé, le *Serveur* doit retourner le *StatusCode* en lieu et place de la *Value*.

Le *Serveur* doit valider les *selectClauses* lorsqu'un *Client* crée ou met à jour l'*EventFilter*. Toutes les éventuelles erreurs qui sont vraies pour tous les *Evénements* possibles sont retournées dans le paramètre *selectClauseResults* décrit au Tableau 132. Le *Serveur* ne doit pas rapporter les erreurs susceptibles de se produire en fonction de l'état du *Serveur* ou du type d'*Evénement*. Par exemple, un *selectClause* qui demande un seul élément d'une matrice produirait toujours une erreur si le *DataType* de l'Attribut est un scalaire. Cependant, même si le *DataType* est une matrice, une erreur pourrait se produire si l'indice demandé n'existe pas pour un *Evénement* particulier. Dans cette dernière situation, le *Serveur* ne rapporterait pas une erreur dans le paramètre *selectClauseResults*.

Le *SimpleAttributeOperand* est utilisé dans la *whereClause* pour sélectionner une valeur qui est partie intégrante d'une expression logique. Ces expressions logiques sont alors utilisées pour déterminer s'il convient de rapporter un *Evénement* particulier au *Client*. Le *Serveur* doit utiliser une valeur null si une erreur se produit lorsqu'une *whereClause* est évaluée pour un *Evénement* particulier. Si un *Value Attribute* a un *StatusCode* incertain ou mauvais qui lui est associé, le *Serveur* doit utiliser une valeur null en lieu et place de la *Value*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tout *FilterOperator* de base contenu dans le Tableau 110 peut être utilisé dans la *whereClause*; cependant, seul l'OfType *FilterOperator* du Tableau 111 est autorisé.

Le *Serveur* doit valider les *whereClauses* lorsqu'un *Client* crée ou met à jour l'*EventFilter*. Toutes les éventuelles erreurs de structure dans la construction du filtre et toutes les éventuelles erreurs qui sont vraies pour tous les *Evénements* possibles sont retournées dans le paramètre *whereClauseResult* décrit au Tableau 132. Les erreurs qui pourraient se produire en fonction de l'état du *Serveur* ou de l'*Evénement* ne font pas l'objet d'un rapport.

Les *SubscriptionControlEvents* sont des *Evénements* spéciaux qui sont utilisés pour fournir des informations de commande au *Client*. Ces *Evénements* ne sont édités qu'aux *MonitoredItems* dans la *Souscription* qui a produit le *SubscriptionControlEvent*. Ces *Evénements* contournent la *whereClause*.

Le Tableau 131 définit la structure EventFilter.

Tableau 131 – Structure EventFilter

Nom	Type	Description
EventFilter	structure	
selectClauses []	SimpleAttribute Operand	Liste des valeurs à retourner avec chaque *Evénement* dans une *Notification*. Au moins une clause valide doit être spécifiée. Voir 7.4.4.5 pour la définition de *SimpleAttributeOperand*.
whereClause	ContentFilter	Limiter les *Notifications* aux *Evénements* qui correspondent aux critères définis par ce ContentFilter. La structure ContentFilter est décrite au 7.4. La structure *AttributeOperand* ne peut pas être utilisée dans les *EventFilters*.

Le Tableau 132 définit la structure EventFilterResult.

Tableau 132 – Structure EventFilterResult

Nom	Type	Description
EventFilterResult	structure	
selectClauseResults []	StatusCode	Liste de codes de statut pour les éléments dans la clause select. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments dans le paramètre de demande *selectClauses*. Le *Serveur* retourne null pour les champs *Evénement* indisponibles ou rejetés.
selectClauseDiagnosticInfos []	DiagnosticInfo	Une liste d'informations de diagnostic pour les éléments individuels dans la clause select. La taille et l'ordre de la liste correspondent à la taille et à l'ordre des éléments dans le paramètre de demande *selectClauses*. Cette liste est vide si les informations de diagnostic n'ont pas été demandées dans l'en-tête de demande ou si aucune information de diagnostic n'a été rencontrée dans le traitement des clauses select.
whereClauseResult	ContentFilter Result	Un resultat associé avec le paramètre de demande *whereClause*. Le type *ContentFilterResult* est défini en 7.4.2.

Le Tableau 133 définit des valeurs pour le paramètre selectClauseResults. Les *StatusCodes* communs sont définis au Tableau 166.

Tableau 133 – Codes de résultats EventFilterResult

Id symbolique	Description
Bad_TypeDefinitionInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Le typeId n'est pas le NodeId pour BaseEventType ou un sous-type de celui-ci.
Bad_NodeIdUnknown	Voir le Tableau 166 pour la description de ce code de résultat. Le browsePath est spécifié mais il n'existera jamais dans un Event.
Bad_BrowseNameInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Le browsePath est spécifié et contient un élément null.
Bad_AttributeIdInvalid	Voir le Tableau 166 pour la description de ce code de résultat. Le nœud spécifié par le chemin de navigation ne permettra jamais que l'identificateur d'attribut donné soit retourné.
Bad_IndexRangeInvalid	Voir le Tableau 166 pour la description de ce code de résultat.
Bad_TypeMismatch	Voir le Tableau 166 pour la description de ce code de résultat. L'indexRange est valide mais la valeur de l'Attribut n'est jamais une matrice.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.16.4 AggregateFilter

L'*AggregateFilter* définit la fonction d'agrégats qu'il convient d'utiliser pour calculer les valeurs à retourner. Voir la CEI 62541-13 pour les détails sur les fonctions d'agrégats possibles. Il spécifie un startTime du premier agrégat à calculer. Le samplingInterval des MonitoringAttributes (voir 7.15) définit la manière dont il convient que le serveur échantillonne la source de données sous-jacente. Le processingInterval spécifie la taille d'une période où l'agrégat est calculé. La queueSize des MonitoringAttributes spécifie le nombre de valeurs traitées qu'il convient de conserver.

L'objectif de l'AggregateFilter n'est pas de lire des données historiques; pour ce faire, il convient d'utiliser le service HistoryRead. Cependant, il est permi de mettre le startTime à une heure qui se situe dans le passé lorsqu'elles sont reçues par le serveur. Il convient que le nombre d'agrégats à calculer dans le passé ne dépasse pas la queueSize définie dans les MonitoringAttributes car les valeurs supérieures à la queueSize seraient directement rejetées et ne seraient jamais retournées au client.

Le startTime et le processingInterval peuvent être révisés par le serveur, mais il convient que le startTime reste dans les mêmes limites (startTime + revisedProcessingInterval * n = revisedStartTime). Ce comportement simplifie l'accès aux valeurs historiques des agrégats qui utilisent les mêmes limites en appelant le service HistoryRead. Le paramètre extensible AggregateFilterResult est utilisé pour retourner la valeur révisée pour l'AggregateFilter.

Certains systèmes sous-jacents peuvent sonder les données et produire plusieurs échantillons avec la même valeur. D'autres systèmes peuvent seulement rapporter les changements apportés aux valeurs. La définition pour chaque type d'agrégat explique la manière de gérer ces deux scénarios différents.

Le *MonitoredItem* rapporte uniquement les valeurs pour des intervalles qui sont achevés lorsque la minuterie d'édition expire. Les données non utilisées sont reportées et utilisées pour calculer une valeur retournée dans la prochaine édition.

Le *ServerTimestamp* pour chaque intervalle doit être l'heure de la fin de l'intervalle de traitement.

L'AggregateFilter est défini au Tableau 134.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 134 – Structure AggregateFilter

Nom	Type	Description
AggregateFilter	structure	
startTime	UTCTime	Début de la période pour calculer l'agrégat la première fois. La taille de chaque période utilisée pour calculer l'agrégat est définie par le samplingInterval des MonitoringAttributes (voir 7.15).
aggregateType	NodeId	Le NodeId de l'objet *HistoryAggregate* qui indique la liste des *Aggregates* à utiliser pour récupérer les données traitées. Voir CEI 62541-13 pour les détails.
processingInterval	Duration	La période à utiliser pour calculer l'agrégat.
aggregateConfiguration	Aggregate Configuration	Ce paramètre permet aux Clients de passer outre les paramétrages de configuration des *Aggregate* fournis par l'objet *AggregateConfiguration*, élément surveillé par élément surveillé. Voir CEI 62541-13 pour les détails pour plus d'informations sur les configurations des *Aggregate*. Si le *Serveur* ne prend pas en charge la capacité de passer outre aux paramétrages de configuration des *Aggregate*, il doit retourner un *StatusCode* Bad_AggregateListMismatch.
useSeverCapabilities Defaults	Boolean	Si value = TRUE utiliser les paramétrages de configuration des agrégats tels qu'indiqués par l'objet AggregateConfiguration. Si value = FALSE utiliser les paramétrages de configuration tels qu'indiqués dans les paramétrages AggregateConfiguration suivants. La valeur par défaut est TRUE.
treatUncertainAsBad	Boolean	Comme décrit dans la CEI 62541-13.
percentDataBad	Byte	Comme décrit dans la CEI 62541-13.
percentDataGood	Byte	Comme décrit dans la CEI 62541-13.
steppedSloped Extrapolation	Boolean	Comme décrit dans la CEI 62541-13.

L'AggregateFilterResult définit l'AggregateFilter révisé que le serveur peut retourner lorsqu'un AggregateFilter est défini pour un MonitoredItem dans les services CreateMonitoredItems ou ModifyMonitoredItems. L'AggregateFilterResult est défini au Tableau 135.

Tableau 135 – Structure AggregateFilterResult

Nom	Type	Description
AggregateFilterResult	structure	
revisedStartTime	UTCTime	L'intervalle StartTime réel que le *Serveur* doit utiliser. Cette valeur est basée sur plusieurs facteurs, notamment les capacités du serveur à accéder aux données historiques. Il convient que le revisedStartTime reste dans les mêmes limites que le startTime (startTime + samplingInterval * n = revisedStartTime).
revisedProcessingInterval	Duration	Le processingInterval réel que le *Serveur* doit utiliser. Le revisedProcessingInterval doit être au moins deux fois le revisedSamplingInterval pour le MonitoredItem.

7.17 Mode de surveillance (MonitoringMode)

Le *MonitoringMode* est une énumération qui spécifie si l'échantillonnage et la production de rapports sont activés ou désactivés pour un *MonitoredItem*. La valeur du paramètre d'édition activé pour une *Souscription* n'a pas d'incidence sur la valeur du mode de surveillance pour un *MonitoredItem* de *Souscription*. Les valeurs de ce paramètre sont définies au Tableau 136.

Tableau 136 – Valeurs de MonitoringMode

Valeur	Description
DISABLED_0	L'élément surveillé n'est ni échantillonné ni évalué, et les *Notifications* ne sont ni générées ni placées en file d'attente. La production de rapports relatifs aux *Notifications* est désactivée.
SAMPLING_1	L'élément surveillé est échantillonné et évalué, et les *Notifications* sont générées et placées en file d'attente. La production de rapports relatifs aux *Notifications* est désactivée.
REPORTING_2	L'élément surveillé est échantillonné et évalué, et les *Notifications* sont générées et placées en file d'attente. La production de rapports relatifs aux *Notifications* est activée.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.18 Paramètres des attibuts d'un nœud (NodeAttributes)

7.18.1 Vue générale

Le *Service AddNodes* permet de spécifier les *Attributs* pour les *Noeuds* à ajouter. Le *NodeAttributes* est un paramètre extensible dont la structure dépend du type de l'*Attribut* ajouté. Il identifie la *NodeClass* qui définit la structure des *Attributs* qui suivent. Les *parameterTypeIds* sont déjà définis au Tableau 137. Le type *ExtensibleParamter* est défini en 7.11.

Tableau 137 –TypeIds du paramètre NodeAttributes

Id symbolique	Description
ObjectAttributes	Définit les *Attributs* pour l'*Object NodeClass*.
VariableAttributes	Définit les *Attributs* pour la *Variable NodeClass*.
MethodAttributes	Définit les *Attributs* pour la *Method NodeClass*.
ObjectTypeAttributes	Définit les *Attributs* pour l'*ObjectType NodeClass*.
VariableTypeAttributes	Définit les *Attributs* pour la *VariableType NodeClass*.
ReferenceTypeAttributes	Définit les *Attributs* pour la *ReferenceType NodeClass*.
DataTypeAttributes	Définit les *Attributs* pour la *DataType NodeClass*.
ViewAttributes	Définit les *Attributs* pour la Vue *NodeClass*.

Le Tableau 138 définit le masque de bits utilisé dans les paramètres NodeAttributes pour spécifier les *Attributs* qui sont établis par le *Client*.

Tableau 138 – Masque de bits pour les attributs spécifiés

Champ	Bit	Description
AccessLevel	0	Indique si l'attribut AccessLevel est positionné.
ArrayDimensions	1	Indique si l'attribut ArrayDimensions est positionné.
Reserved	2	Réservé pour être cohérent avec le WriteMask défini dans la CEI 62541-3.
ContainsNoLoops	3	Indique si l'attribut ContainsNoLoops est positionné.
DataType	4	Indique si l'attribut DataType est positionné.
Description	5	Indique si l'attribut Description est positionné.
DisplayName	6	Indique si l'attribut DisplayName est positionné.
EventNotifier	7	Indique si l'attribut EventNotifier est positionné.
Executable	8	Indique si l'attribut Executable est positionné.
Historizing	9	Indique si l'attribut Historizing est positionné.
InverseName	10	Indique si l'attribut InverseName est positionné.
IsAbstract	11	Indique si l'attribut IsAbstract est positionné.
MinimumSamplingInterval	12	Indique si l'attribut MinimumSamplingInterval est positionné.
Reserved	13	Réservé pour être cohérent avec le WriteMask défini dans la CEI 62541-3.
Reserved	14	Réservé pour être cohérent avec le WriteMask défini dans la CEI 62541-3.
Symmetric	15	Indique si l'attribut Symmetric est positionné.
UserAccessLevel	16	Indique si l'attribut UserAccessLevel est positionné.
UserExecutable	17	Indique si l'attribut UserExecutable est positionné.
UserWriteMask	18	Indique si l'attribut UserWriteMask est positionné.
ValueRank	19	Indique si l'attribut ValueRank est positionné.
WriteMask	20	Indique si l'attribut WriteMask est positionné.
Value	21	Indique si l'attribut Value est positionné.
Reserved	22:32	Réservé pour usage futur. Doit toujours être zéro.

7.18.2 Paramètre d'ObjectAttributes

Le Tableau 139 définit le paramètre *ObjectAttributes*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 139 – ObjectAttributes

Nom	Type	Description
ObjectAttributes	structure	Définit les *Attributs* pour l'*Object NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est positionné à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
eventNotifier	Byte	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*

7.18.3 Paramètre de VariableAttributes

Le Tableau 140 définit le paramètre *VariableAttributes*.

Tableau 140 – VariableAttributes

Nom	Type	Description
VariableAttributes	structure	Définit les *Attributs* pour la *Variable NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
value	Défini par le *DataType Attribute*	Voir la CEI 62541-3 pour la description de cet *Attribut*.
dataType	NodeId	Voir la CEI 62541-3 pour la description de cet *Attribut*.
valueRank	Int32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
arrayDimensions	UInt32 []	Voir la CEI 62541-3 pour la description de cet *Attribut*.
accessLevel	Byte	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userAccessLevel	Byte	Voir la CEI 62541-3 pour la description de cet *Attribut*.
minimumSamplingInterval	Duration	Voir la CEI 62541-3 pour la description de cet *Attribut*.
historizing	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.4 Paramètre de MethodAttributes

Le Tableau 141 définit le paramètre *MethodAttributes*.

Tableau 141 – MethodAttributes

Nom	Type	Description
BaseAttributes	structure	Définit les *Attributs* pour la *Method NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
executable	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userExecutable	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.5 Paramètre d'ObjectTypeAttributes

Le Tableau 142 définit le paramètre *ObjectTypeAttributes*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 142 – ObjectTypeAttributes

Nom	Type	Description
ObjectTypeAttributes	structure	Définit les *Attributs* pour l'*ObjectType NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
isAbstract	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.6 Paramètre de VariableTypeAttributes

Le Tableau 143 définit le paramètre *VariableTypeAttributes*.

Tableau 143 – VariableTypeAttributes

Nom	Type	Description
VariableTypeAttributes	structure	Définit les *Attributs* pour la *VariableType NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
value	Défini par le *DataType Attribute*	Voir la CEI 62541-3 pour la description de cet *Attribut*.
dataType	NodeId	Voir la CEI 62541-3 pour la description de cet *Attribut*.
valueRank	Int32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
arrayDimensions	UInt32 []	Voir la CEI 62541-3 pour la description de cet *Attribut*.
isAbstract	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.7 Paramètre de ReferenceTypeAttributes

Le Tableau 144 définit le paramètre *ReferenceTypeAttributes*.

Tableau 144 – ReferenceTypeAttributes

Nom	Type	Description
ReferenceTypeAttributes	structure	Définit les *Attributs* pour la *ReferenceType NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
isAbstract	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
symmetric	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
inverseName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.8 Paramètre de DataTypeAttributes

Le Tableau 145 définit le paramètre *DataTypeAttributes*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 145 – DataTypeAttributes

Nom	Type	Description
DataTypeAttributes	structure	Définit les *Attributs* pour la *DataType NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
isAbstract	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.18.9 Paramètre de ViewAttributes

Le Tableau 146 définit le paramètre *ViewAttributes*.

Tableau 146 – ViewAttributes

Nom	Type	Description
ViewAttributes	structure	Définit les *Attributs* pour la *View NodeClass*.
specifiedAttributes	UInt32	Un masque de bits qui indique les champs qui contiennent des valeurs valides. Un champ doit être ignoré si le bit correspondant est mis à 0. Les valeurs des bits sont définies au Tableau 138.
displayName	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
Description	LocalizedText	Voir la CEI 62541-3 pour la description de cet *Attribut*.
containsNoLoops	Boolean	Voir la CEI 62541-3 pour la description de cet *Attribut*.
eventNotifier	Byte	Voir la CEI 62541-3 pour la description de cet *Attribut*.
writeMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.
userWriteMask	UInt32	Voir la CEI 62541-3 pour la description de cet *Attribut*.

7.19 Paramètres des données de notification (NotificationData)

7.19.1 Vue générale

La structure *NotificationMessage* utilisée dans le jeu de *Services Souscription* permet de spécifier différents types de *NotificationData*. Le *NotificationData* est un paramètre extensible dont la structure dépend du type de *Notification* envoyée. Ce paramètre est défini au Tableau 147. D'autres types peuvent être définis par des parties supplémentaires de la série de normes ou par d'autres spécifications basées sur OPC UA. Le type *ExtensibleParamter* est défini en 7.11.

Il peut y avoir plusieurs notifications pour un seul MonitoredItem dans une seule structure NotificationData. Lorsque cela se produit, le *Serveur* doit assurer que les notifications apparaissent dans l'ordre dans lequel elles étaient placées en file d'attente dans le MonitoredItem. Ces notifications peuvent ne pas apparaître dans un bloc contigu.

Tableau 147 – TypeIds du paramètre NotificationData

Id symbolique	Description
DataChange	Paramètre de donnée *Notification* utilisée pour les *Notifications* de changements de donnée.
Evénement	Paramètre de donnée *Notification* utilisée pour les *Event Notifications*.
StatusChange	Paramètre de donnée *Notification* utilisée pour les *Notifications* de changements de statut d'abonnement.

7.19.2 Paramètre de DataChangeNotification

Le Tableau 148 définit le paramètre *NotificationData* utilisé pour des notifications de changements de donnée. Cette structure contient les éléments de données surveillés qui sont à rapporter. Les éléments de données surveillés sont rapportés dans deux conditions:

a) si le *MonitoringMode* est mis à REPORTING et si un changement est apporté à une valeur ou à son statut (représenté par son *StatusCode*) est détecté;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

b) si le *MonitoringMode* est mis à SAMPLING, le *MonitoredItem* est relié à un élément déclencheur qui déclenche.

Voir 5.12 pour une description du jeu de *Services MonitoredItem* et en particulier le *MonitoringItemModel* et le *TriggeringModel*.

Après la création d'un *MonitoredItem*, la valeur ou le statut courant(e) de l'Attribut surveillé doit être placé en file d'attente sans appliquer de filtre. Si la valeur courante n'est pas disponible après le premier intervalle d'échantillonnage, la première *Notification* doit être placée en file d'attente dans la source de données après récupération de la valeur ou du statut initial.

Tableau 148 – DataChangeNotification

Nom	Type	Description
DataChangeNotification	structure	Donnée de *Notification* de changements de donnée.
monitoredItems []	MonitoredItem Notification	La liste des *MonitoredItems* pour lesquels un changement a été détecté.
clientHandle	IntegerId	Descripteur fourni par le *Client* pour le *MonitoredItem*. Le type *IntegerId* est défini en 7.13
value	DataValue	Le *StatusCode*, la valeur et le(s) marqueur(s) temporel(s) de l'*Attribut* surveillé en fonction de configuration d'échantillonnage et de mise en file d'attente. Si le *StatusCode* indique une erreur, la valeur et le(s) marqueur(s) temporel(s) sont à ignorer. Si les changements détectés n'ont pas tous été retournés depuis que le tampon de file d'attente du *Serveur* pour les *MonitoredItem* a atteint sa limite et a dû purger les données, le bit *Overflow* dans les *DataValue InfoBits* du *statusCode* est positionné. *DataValue* est un type commun défini en 7.7.
diagnosticInfos []	DiagnosticInfo	Liste d'informations de diagnostic. La taille et l'ordre de cette liste correspondent à la taille et à l'ordre du paramètre de demande *monitoredItem*. Il y a une entrée dans cette liste pour chaque Noeud contenu dans le paramètre *monitoredItem*. Cette liste est vide si les informations de diagnostic n'avaient pas été demandées ou ne sont disponibles pour aucun des *MonitoredItems*. *DiagnosticInfo* est un type commun défini en 7.8

7.19.3 Paramètre d'EventNotificationList

Le Tableau 149 définit le paramètre *NotificationData* utilisé pour les *EventNotifications*.

L'EventNotificationList définit une structure de tableau qui est utilisée pour retourner des champs d'*Evénement* à un *Abonnement de Client (Client Subscription)*. La structure a la forme d'un tableau constitué d'un ou plusieurs *Evénements*, contenant chacun une matrice d'un ou plusieurs champs. La sélection et l'ordre des champs retournés pour chaque *Evénement* sont identiques à ceux du paramètre sélectionné de l'*EventFilter*.

Tableau 149 – EventNotificationList

Nom	Type	Description
EventNotificationList	structure	Données de *Notification* d'événement.
events []	EventFieldList	La liste des *Events* distribués.
clientHandle	IntegerId	Descripteur fourni par le *Client* pour le *MonitoredItem*. Le type *IntegerId* est défini en 7.13.
eventFields []	BaseDataType	Liste de champs d'*Event* sélectionnés. Ceux-ci doivent être en correspondance univoque avec les champs sélectionnés dans l'*EventFilter*. Le paragraphe 7.16.3 spécifie la manière dont le *Serveur* doit traiter les conditions d'erreur.

7.19.4 Paramètre de StatusChangeNotification

Le Tableau 150 définit le paramètre *NotificationData* utilisé pour un *StatusChangeNotification*.

La *StatusChangeNotification* informe le client d'un changement de statut d'une *Souscription*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 150 – StatusChangeNotification

Nom	Type	Description
StatusChangeNotification	structure	Données de *Notification* d'événement.
Status	StatusCode	Le *StatusCode* qui indique le changement de statut.
diagnosticInfo	DiagnosticInfo	*DiagnosticInformation* pour le changement de statut.

7.20 Message de notification (NotificationMessage)

Les composantes de ce paramètre sont définies au Tableau 151.

Tableau 151 – NotificationMessage

Nom	Type	Description
NotificationMessage	structure	Le *Message* qui contient une ou plusieurs *Notifications*.
sequenceNumber	Counter	Le numéro de séquence du *NotificationMessage*.
publishTime	UTCTime	L'heure à laquelle ce *Message* a été envoyé au *Client*. Si ce *Message* est retransmis au *Client*, ce paramètre contient l'heure à laquelle il a été transmis la première fois au *Client*.
notificationData []	Extensible Parameter NotificationData	La liste des *structures NotificationData.* Le type de paramètre *NotificationData* est un type de paramètre extensible spécifié en 7.19. Il spécifie les types des *Notifications* qui peuvent être envoyées. Le type *ExtensibleParameter* est spécifié en 7.11. Il convient de regrouper les Notifications du même type dans un seul élément NotificationData. Si un Abonnement contient des *MonitoredItems* pour des événements et des données, il convient que cette matrice n'ait pas plus de deux éléments. Si la *Souscription* contient des *MonitoredItems* uniquement pour des données ou uniquement pour des événements, il convient que la taille de la matrice soit toujours 1 pour cette *Souscription*.

7.21 Plage numérique (NumericRange)

Ce paramètre est défini au Tableau 152. Une définition BNF formelle de la plage numérique peut être trouvée à l'Article A.3.

La syntaxe pour la chaîne contient l'une des deux constructions suivantes. La première construction est la représentation sous forme de chaîne d'un nombre entier individuel. Par exemple, “6” est valide, mais “6,0” et “3,2” ne le sont pas. Les valeurs minimales et maximales qui peuvent être exprimées sont définies par l'utilisation de ce paramètre, et non par la définition du type de ce paramètre. La seconde construction est une plage représentée par deux nombres entiers séparés par le caractère deux-points (“:”). Le premier nombre entier doit toujours avoir une valeur inférieure à celle du second. Par exemple, “5:7” est valide, alors que “7:5” et “5:5” ne le sont pas. Les valeurs minimales et maximales qui peuvent être exprimées par ces nombres entiers sont définies par l'utilisation de ce paramètre, et non par la définition du type de ce paramètre. Aucun autre caractère, y compris les caractères espace, n'est autorisé.

Les matrices multidimensionnelles peuvent être indexées en spécifiant une plage pour chaque dimension séparée par une ‘,’. Par exemple, un bloc 2x2 dans une matrice 4x4 pourrait être sélectionné avec la plage “1:2,0:1”. Un élément individuel de la matrice multidimensionnelle peut être sélectionné en spécifiant un seul nombre au lieu d'une plage. Par exemple, “1,1” sélectionne l'élément [1,1] dans une matrice à deux dimensions.

Les dimensions sont spécifiées dans l'ordre de leur apparition dans l'attribut *ArrayDimensions*. Toutes les dimensions doivent être spécifiées pour qu'un *NumericRange* soit valide.

Tous les indices commencent à 0. La valeur maximale pour un indice quelconque est inférieure de 1 à la longueur de la dimension.

Lors de la lecture d'une valeur, les indices ne peuvent pas spécifier une plage qui est dans les limites de la matrice. Le *Serveur* doit retourner un résultat partiel si certains éléments

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

existent dans la plage. Le *Serveur* doit retourner un *Bad_OutOfRange* si aucun élément n'existe dans la plage.

Lors de l'écriture d'une valeur, la taille de la matrice doit correspondre à la taille spécifiée par la *NumericRange*. Le *Serveur* doit retourner une erreur s'il ne peut pas écrire tous les éléments spécifiés par le *Client*.

La *NumericRange* peut aussi être utilisée pour spécifier des sous-chaînes pour les valeurs *ByteString* et *String*. Les matrices de valeurs *ByteString* et *String* sont traitées comme des matrices à deux dimensions dans lesquelles l'indice final spécifie la plage de la sous-chaîne dans la valeur *ByteString* ou *String*. La valeur complète *ByteString* ou *String* est sélectionnée si l'indice final est omis.

Tableau 152 – NumericRange

Nom	Type	Description
NumericRange	String	Un nombre ou une plage numérique. Une chaîne null indique que ce paramètre n'est pas utilisé.

7.22 Donnée de demande (QueryDataSet)

Les composantes de ce paramètre sont définies au Tableau 153.

Tableau 153 – QueryDataSet

Nom	Type	Description
QueryDataSet	structure	Données relatives à un Noeud retournées dans une réponse Query.
nodeId	ExpandedNodeId	Le *NodeId* pour la description de ce Noeud.
typeDefinitionNode	ExpandedNodeId	Le *NodeId* pour la définition de type pour la description de ce Noeud.
values []	BaseDataType	Valeurs pour les *Attributs* sélectionnés. L'ordre des éléments retournés correspond à l'ordre des éléments demandés. Il y a une entrée pour chaque élément demandé pour le *TypeDefinitionNode* donné qui correspond à l'instance sélectionnée; cela inclut tous les éventuels nœuds connexes qui avaient été spécifiés en utilisant le chemin relatif partant du *TypeDefinitionNode* de l'instance sélectionnée. Si aucune valeur n'a été trouvée pour un élément demandé donné, une valeur null est retournée pour l'élément en question. Si plusieurs valeurs existent pour un élément demandé, une matrice de valeurs est retournée. Si l'élément demandé est une référence, une *ReferenceDescription* ou une matrice de *ReferenceDescriptions* est retournée pour l'élément en question.

7.23 Lecture de l'identifiant d'une valeur (ReadValueId)

Les composantes de ce paramètre sont définies au Tableau 154.

Tableau 154 – ReadValueId

Nom	Type	Description
ReadValueId	structure	Identificateur pour un élément à lire ou à surveiller.
nodeId	NodeId	*NodeId* d'un Noeud.
AttributeID	IntegerId	Identificateur de l'*Attribut*. Il doit être un identificateur d'*Attribut* valide. L'*IntegerId* est défini en 7.13. Les IntegerIds pour les attributs sont définis dans la CEI 62541-6.
indexRange	NumericRange	Ce paramètre est utilisé pour identifier un élément simple d'une matrice ou une plage simple d'indices pour les matrices. Si une plage d'éléments est spécifiée, les valeurs sont retournées sous forme de composé. Le premier élément est identifié par l'indice 0 (zéro). Le type *NumericRange* est défini en 7.21. Ce paramètre est null si l'*Attribut* spécifié n'est pas une matrice. Cependant, si l'*Attribut* spécifié est une matrice et ce paramètre est null, tous les éléments doivent être inclus dans la plage.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
dataEncoding	QualifiedName	Ce paramètre spécifie le *BrowseName* du *DataTypeEncoding* qu'il convient que le *Serveur* utilise pour retourner l'attribut *Value* d'une *Variable*. C'est une erreur de spécifier ce paramètre pour d'autres *Attributs*. Un *Client* peut découvrir les *DataTypeEncoding*s qui sont disponible en suivant la *Référence HasEncoding* à partir du Noeud *DataType* pour une *Variable*. OPC UA définit des *BrowseNames* que les *Serveurs* doivent reconnaître même si les *Nodes DataType* ne sont pas visibles dans l'Espace d'Adresses du *Serveur*. Ces *BrowseNames* sont: DefaultBinary The default or native binary (or non-XML) encoding. DefaultXML The default XML encoding. Chaque *DataType* doit prendre en charge au moins l'un de ces codages. Il convient que les *DataTypes* qui n'ont pas un véritable codage binaire (par exemple ils ont seulement un codage de texte non-XML) utilisent le nom *DefaultBinary* pour identifier le codage qui est considéré être le codage non-XML par défaut. Les *DataTypes* qui prennent en charge au moins un codage XML doivent identifier l'un des codages comme étant le codage DefaultXML. D'autres organismes de normalisation peuvent définir d'autres codages de données bien connus qui pourraient être pris en charge. Si ce paramètre n'est pas spécifié, le *Serveur* doit choisir soit le codage DefaultBinary, soit le codage *DefaultXML* en fonction du codage de *Message* (voir la CEI 62541-6) qui est utilisé pour la *Session*. Si le *Serveur* ne prend pas en charge le codage qui correspond au codage de *Message*, le *Serveur* doit choisir le codage par défaut qu'il prend effectivement en charge. Si ce paramètre est spécifié pour un *MonitoredItem*, le *Serveur* doit mettre le bit *StructureChanged* dans le *StatusCode* (voir 7.33) si le *DataTypeEncoding* change. Le *DataTypeEncoding* change si la *DataTypeVersion* de la *DataTypeDescription* ou du *DataTypeDictionary* associé au *DataTypeEncoding* change.

7.24 Description d'une référence (ReferenceDescription)

Les composantes de ce paramètre sont définies au Tableau 155.

Tableau 155 – ReferenceDescription

Nom	Type	Description
ReferenceDescription	structure	Le paramètre de référence retourné pour le *Browse Service* (service navigation).
referenceTypeId	NodeId	*NodeId* du *ReferenceType* qui définit la *Référence*.
isForward	Boolean	Si la valeur est TRUE, le *Serveur* a suivi une *Référence* directe (forward Reference). Si la valeur est FALSE, le *Serveur* a suivi une *Référence* inverse (inverse Reference).
nodeId	Expanded NodeId	*NodeId* du *TargetNode* tel qu'affecté par le *Serveur* identifié par l'indice de *Serveur*. Le type *ExpandedNodeId* est défini en 7.10. Si le *server*Index indique que le *TargetNode* est un Noeud distant, le *nodeId* doit contenir l'URI d'espace de noms absolu. Si le *TargetNode* est un Noeud local, le *nodeId* doit contenir l'indice d'espace de noms.
browseName[a)]	QualifiedName	Le *BrowseName* du *TargetNode*.
displayName	LocalizedText	Le *DisplayName* du *TargetNode*.
nodeClass[a)]	NodeClass	*NodeClass* du *TargetNode*.
typeDefinition[a)]	Expanded NodeId	*NodeId* de définition de type du *TargetNode*. Les définitions de type sont disponibles uniquement pour les *NodeClasses Object* et *Variable*. Pour toutes les autres *NodeClasses*, un NodeId null doit être retourné.
[a)] Si l'indice du *Serveur* indique que le *TargetNode* est un Noeud distant, le *browseName*, la nodeClass et la typeDefinition du *TargetNode* peuvent être null ou vides. S'ils ne le sont pas, ils peuvent ne pas être à jour car le *Serveur* local peut ne pas surveiller en permanence le *Serveur* distant pour détecter les changements.		

7.25 Chemin relatif (RelativePath)

Les composantes de ce paramètre sont définies au Tableau 156.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 156 – RelativePath

Nom	Type	Description
RelativePath	structure	Définit une séquence de *Références* et de *BrowseNames* à suivre.
elements []	RelativePath Element	Une séquence de *Références* et de *BrowseNames* à suivre. Chaque élément dans la séquence est traité en trouvant les cibles et ensuite en utilisant ces cibles comme nœuds de départ pour le prochain élément. Les cibles de l'élément final sont la cible du *RelativePath*.
referenceTypeId	NodeId	Le type de référence à suivre à partir du nœud courant. Le chemin courant ne peut pas être suivi plus loin si le referenceTypeId n'est pas disponible sur l'instance du nœud.
isInverse	Boolean	Indique s'il convient de suivre la *Référence* inverse. La référence inverse est suivie si cette valeur est TRUE.
includeSubtypes	Boolean	Indique s'il convient de suivre les sous-types du *ReferenceType*. Les sous-types sont inclus si cette valeur est TRUE.
targetName	QualifiedName	Le *BrowseName* du nœud cible L'élément final peut avoir un *targetName* vide. Dans cette situation, toutes les cibles des références identifiées par le referenceTypeId sont les cibles du *RelativePath*. Le *targetName* doit être spécifié pour tous les autres éléments. Le chemin courant ne peut pas être suivi si aucune cible avec le *BrowseName* spécifié n'existe.

Une *RelativePath* peut être appliqué à tout Noeud de départ. Les cibles du *RelativePath* sont le jeu de Noeuds qui sont trouvés en suivant séquentiellement les éléments dans le *RelativePath*.

Un format texte pour la *RelativePath* peut être trouvé dans l'Article A.2. Ce format est utilisé dans des exemples qui expliquent les *Services* qui se servent de la structure du *RelativePath*.

7.26 Entête d'une demande (RequestHeader)

Les composantes de ce paramètre sont définies au Tableau 157.

Tableau 157 – RequestHeader

Nom	Type	Description
RequestHeader	structure	Paramètres communs pour toutes les demandes présentées sur une *Session*.
authenticationToken	Session AuthenticationToken	L'identificateur de *Session* secret utilisé pour vérifier que la demande est associée à la *Session.* Le type *SessionAuthenticationToken* est défini en 7.29.
timestamp	UTCTime	L'heure à laquelle le *Client* a envoyé la demande.
requestHandle	IntegerId	Un *requestHandle* associé à la demande. Ce descripteur défini par le client peut être pour annuler la demande. Il est également retourné dans la réponse.
returnDiagnostics	UInt32	Un masque de bits qui identifie les types de diagnostics spécifiques au vendeur devant être retournés dans les paramètres de réponse *diagnosticInfo*. La valeur du paramètre peut être constituée de zéro, une ou plusieurs des valeurs ci-après. Aucune valeur n'indique que les diagnostics ne sont pas à retourner. Valeur de bit — Diagnostic à retourner 0x0000 0001 ServiceLevel / SymbolicId 0x0000 0002 ServiceLevel / LocalizedText 0x0000 0004 ServiceLevel / AdditionalInfo 0x0000 0008 ServiceLevel / *StatusCode* interne 0x0000 0010 ServiceLevel / Diagnostics internes 0x0000 0020 OperationLevel / SymbolicId 0x0000 0040 OperationLevel / LocalizedText 0x0000 0080 OperationLevel / AdditionalInfo 0x0000 0100 OperationLevel / *StatusCode* interne 0x0000 0200 OperationLevel / Diagnostics internes Chacune de ces valeurs est composée de deux composants, à savoir *level* (niveau) et *type*, comme décrit ci-dessous. Si aucun n'est demandé, comme indiqué par une valeur 0, ou si aucune information de diagnostic n'a été rencontrée dans le traitement de la demande, les informations de diagnostic ne sont pas retournées. *Level*: ServiceLevel retourner des diagnostics dans le *diagnosticInfo* du *Service*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Nom	Type	Description
		OperationLevel retourner des diagnostics dans le *diagnosticInfo* défini pour des opérations individuelles demandées par e *Service*. *Type*: SymbolicId retourner un identificateur symbolique qualifié pour un espace de noms pour une erreur ou une condition. La longueur maximale de cet identificateur est de 32 caractères. LocalizedText retourner jusqu'à 256 octets de texte localisé qui décrit l'identificateur symbolique. AdditionalInfo retourner une chaîne d'octets qui contient des informations de diagnostic supplémentaires, telles qu'une image mémoire. Le format de cette chaîne d'octets est spécifique au vendeur et peut dépendre du type d'erreur ou de condition rencontrée. InnerStatusCode retourner le *StatusCode* interne associé à l'opération ou au *Service*. InnerDiagnostics retourner les informations de diagnostic internes associées à l'opération ou au *Service*. Le contenu de la structure d'informations de diagnostic internes est déterminé par d'autres bits dans le masque. A remarquer que le positionnement de ce bit peut induire que plusieurs niveaux de structures d'informations de diagnostic imbriquées soient retournés.
auditEntryId	String	Un identificateur qui identifie l'entrée de journal d'audits de sécurité du *Client* associée à cette demande. Une valeur de chaîne vide signifie que ce paramètre n'est pas utilisé. L'*AuditEntryId* contient typiquement l'initiateur de l'action et l'endroit où elle a été initiée. L'*AuditEventId* est inclus dans l'*AuditEvent* pour permettre à des lecteurs humains de corréler un *Event* à une action initiatrice. Plus de détails des mécanismes d'*Audit* sont définis en 6.2 et en CEI 62541-3.
timeoutHint	UInt32	Cette temporisation en millisecondes est utilisée dans la *Pile Communication* côté *Client* pour établir la temporisation sur une base appel par appel. Pour un *Serveur*, cette temporisation est seulement une suggestion et peut être utilisée pour annuler des opérations de longue durée pour libérer des ressources. Si le serveur détecte une temporisation, il peut annuler l'opération en envoyant le résultat de *Service Bad_Timeout*. Il convient que le *Serveur* attende au minimum la temporisation après avoir reçu la demande avant d'annuler l'opération. La valeur de 0 indique l'absence de temporisation.
additionalHeader	Extensible Parameter AdditionalHeader	Réservé pour usage futur. Il convient que les applications qui ne comprennent pas l'en-tête l'ignorent.

7.27 Entête d'une réponse (ResponseHeader)

Les composantes de ce paramètre sont définies au Tableau 158.

Tableau 158 – ResponseHeader

Nom	Type	Description
ResponseHeader	structure	Paramètres communs pour toutes les réponses.
timestamp	UTCTime	L'heure à laquelle le *Serveur* a envoyé la réponse.
requestHandle	IntegerId	Le requestHandle donné par le *Client* à la demande.
serviceResult	StatusCode	Résultat défini par l'OPC UA de l'invocation du *Service*. Le type *StatusCode* est défini en 7.33.
serviceDiagnostics	DiagnosticInfo	Informations de diagnostic pour l'invocation du *Service*. Ce paramètre est vide si les informations de diagnostic n'avaient pas été demandées dans l'en-tête de demande. Le type *DiagnosticInfo* est défini en 7.8.
stringTable []	String	Il y a une chaîne dans cette liste pour chaque unique espace de noms, identificateur symbolique, et chaîne de texte localisé contenue dans l'ensemble de tous les paramètres d'informations de diagnostic contenus dans la réponse (voir 7.8). Chacun est identifié dans ce tableau par son indice à base zéro.
additionalHeader	Extensible Parameter AdditionalHeader	Réservé pour usage futur. Il convient que les applications qui ne comprennent pas l'en-tête l'ignorent.

7.28 Service en défaut ServiceFault

Les composantes de ce paramètre sont définies au Tableau 159.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le paramètre *ServiceFault* est retourné au lieu du message de réponse du Service lorsqu'une erreur au niveau service se produit. Il convient que le *requestHandle* dans le *ResponseHeader* soit mis à ce qui avait été fourni dans le *RequestHeader* même si ces valeurs n'étaient pas valides. Le niveau de diagnostic retourné dans le *ResponseHeader* est spécifié par le paramètre *returnDiagnostics* dans le *RequestHeader*.

L'utilisation exacte de ce paramètre dépend des mises en application définies dans la CEI 62541-6.

Tableau 159 – ServiceFault

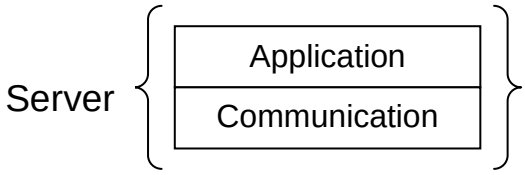
Nom	Type	Description
ServiceFault	structure	Une réponse d'erreur envoyée lorsqu'une erreur de niveau service se produit.
responseHeader	ResponseHeader	Paramètres communs aux réponses (voir 7.27 pour la définition de *ResponseHeader*).

7.29 Autentification du jeton d'une session (SessionAuthenticationToken)

Le type *SessionAuthenticationToken* est un identificateur opaque qui est utilisé pour identifier les demandes associées à une Session particulière. Cet identificateur est utilisé en conjonction avec le *SecureChannelId* ou le *Client Certificate* pour authentifier les messages entrants. Il s'agit de la forme secrète du *sessionId* à usage interne dans les *Applications Client* et *Serveur*.

Un *Serveur* retourne un *SessionAuthenticationToken* dans la réponse *CreateSession*. Le *Client* envoie alors cette valeur avec chaque demande, ce qui permet au *Serveur* de vérifier que l'expéditeur de la demande est le même que l'expéditeur de la demande *CreateSession* d'origine.

Pour les besoins de ce débat, un *Serveur* consiste en une application (code) et une *Pile de Communication* (Communication *Stack*) comme présenté à la Figure 27. La sécurité assurée par le *SessionAuthenticationToken* dépend de la relation de confiance entre l'application *Serveur* et la *Pile de Communication*. La *Pile de Communication* doit être capable de s'assurer de l'expéditeur du message et elle utilise le *SecureChannelId* ou le *Client Certificate* pour identifier l'expéditeur auprès du *Serveur*. Dans ces cas, le *SessionAuthenticationToken* est un identificateur UInt32 qui permet au *Serveur* de distinguer entre des *Sessions* différentes créées par le même expéditeur.



Légende

Anglais	Français
Server	*Serveur*
Application	Application
Communication	Communication

Figure 27 – Couches logiques d'un *Serveur*

Dans certains cas, l'application et la *Pile de Communication* ne peuvent pas échanger d'informations en mode exécution, ce qui signifie que l'application n'aura pas accès au *SecureChannelId* ou au *Certificat* utilisé pour créer le *Canal Sécurisé.* Dans ces cas, l'application doit créer une valeur *ByteString* aléatoire qui a une longueur d'au moins 32 octets. Cette valeur doit être gardée secrète et doit toujours être échangée sur un *Canal Sécurisé* avec le chiffrement activé. L'Administrateur a la responsabilité de s'assurer que le chiffrement est activé. Les *Profiles* dans la CEI 62541-7 peuvent définir des exigences supplémentaires pour un *ByteString SessionAuthenticationToken.*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Il convient que les applications Client et *Serveur* soient écrites de manière à être indépendantes de l'implémentation du *Canal Sécurisé*. Par conséquent, il convient qu'elles traitent toujours le *SessionAuthenticationToken* comme une information secrète, même si cela n'est pas requis lors de l'utilisation de certaines implémentations du *Canal Sécurisé*.

La Figure 28 illustre les informations échangées entre le Client, le *Serveur* et la *Pile de Communication* du *Serveur* lorsque le *Client* obtient un SessionAuthenticationToken. Dans la Figure 28, l'étape GetSecureChannelInfo représente une API qui dépend de l'implémentation de la *Pile de Communication*.

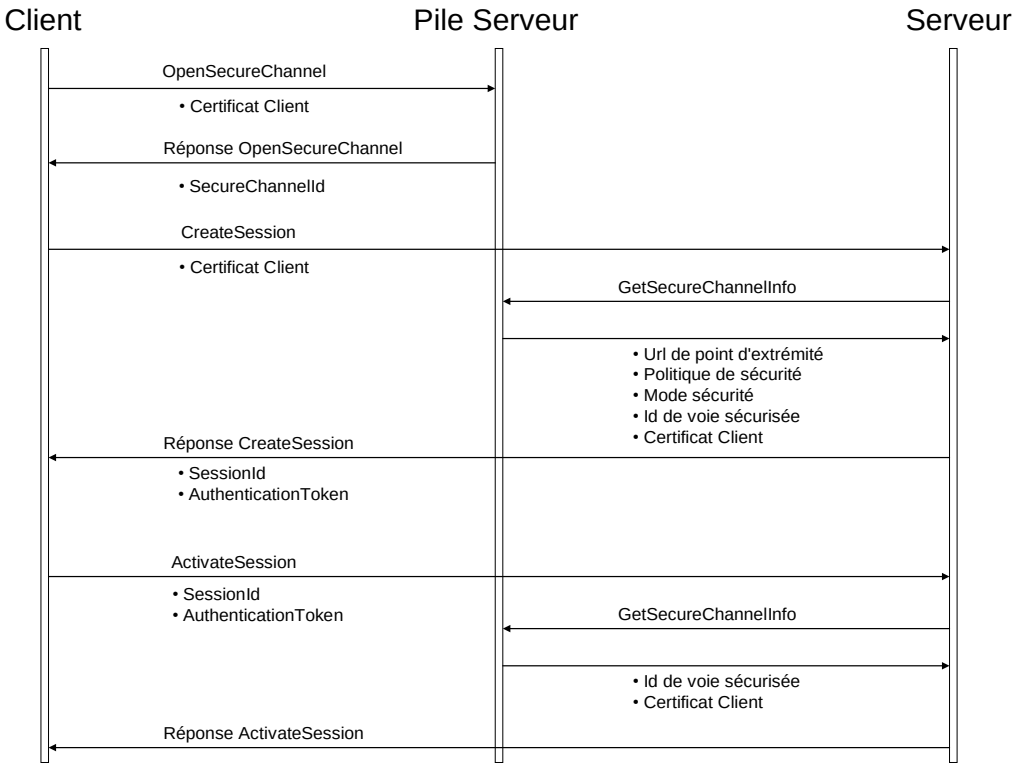


Figure 28 – Obtenir un SessionAuthenticationToken

Le *SessionAuthenticationToken* est un sous-type du type de donnée *NodeId*; cependant, il n'est jamais utilisé pour identifier un Noeud dans l'Espace d'Adresses. Des *Serveurs* peuvent affecter une valeur au *NamespaceIndex*; cependant, sa signification est spécifique au *Serveur*.

7.30 Signature d'une donnée (SignatureData)

Les composantes de ce paramètre sont définies au Tableau 160.

Tableau 160 – SignatureData

Nom	Type	Description
SignatureData	structure	Contient une signature numérique créée avec un Certificat.
signature	ByteString	Il s'agit d'une signature générée par la clé privée associée à un Certificat.
agorithme	String	Une chaîne contenant l'URI de l'algorithme. Les valeurs de la chaîne URI sont des noms définis qui peuvent être utilisés comme partie des profils de sécurité spécifiés dans la CEI 62541-7.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7.31 Certificat d'un logiciel signé (SignedSoftwareCertificate)

Un *SignedSoftwareCertificate* est un *ByteString* contenant un Certificat codé. Le codage d'un *SignedSoftwareCertificate* dépend de la mise en correspondance de la technologie de sécurité et il est complètement défini dans la CEI 62541-6. Le Tableau 161 spécifie les informations qui doivent être contenues dans un *SignedSoftwareCertificate*.

Tableau 161 – SignedSoftwareCertificate

Nom	Type	Description
SignedSoftwareCertificate	structure	Un *SoftwareCertificate* avec une signature créée par l'autorité de certification.
version	String	Un identificateur pour la version du codage du *Certificat*.
serialNumber	ByteString	Un Identificateur unique pour le *Certificat* affecté par son Emetteur.
signatureAlgorithm	String	L'algorithme utilisé pour signer le *Certificat*. La syntaxe de ce champ dépend du codage du *Certificat*.
signature	ByteString	La signature créée par l'Émetteur.
issuer	structure	Un nom qui identifie le *Certificat* de l'Émetteur utilisé pour créer la signature.
validFrom	UTCTime	Le moment où le *Certificat* devient valide.
validTo	UTCTime	Le moment où le *Certificat* expire.
subject	structure	Un nom qui identifie le produit que le *Certificat* décrit. Ce champ doit contenir le *productName* et le *vendorName* issus du *SoftwareCertificate*.
subjectAltName []	structure	Une liste de noms de remplacement pour le produit. Cette liste doit inclure le *productUri* spécifié dans le *SoftwareCertificate*.
publicKey	ByteString	La clé publique associée au *Certificat*.
keyUsage []	String	Spécifie comment la clé de *Certificat* peut être utilisée. Les *SignedSoftwareCertificates* peuvent seulement être utilisés pour créer des signatures numériques et le non-rejet. Le contenu de ce champ dépend du codage du *Certificat*.
softwareCertificate	ByteString	La forme codée XML du *SoftwareCertificate* stockée sous forme de texte UTF8. La CEI 62541-6 décrit la représentation XML pour le *SoftwareCertificate.*

7.32 Certicat d'un logiciel (SoftwareCertificate)

Les composantes de ce paramètre sont définies au Tableau 162.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 162 – SoftwareCertificate

Nom	Type	Description
SoftwareCertificate	structure	Un *Certificat* décrivant un produit.
productName	String	Le nom du produit qui est certifié. Ce champ doit être spécifié.
productUri	String	Un identificateur unique au niveau global pour le produit qui est certifié. Ce champ doit être spécifié.
vendorName	String	Le nom du vendeur responsable du produit. Ce champ doit être spécifié.
vendorProductCertificate	ByteString	La forme codée DER (*Distinguished Encoding Rules*, règles de codage distinctives) du *Certificat* X.509 qui est affecté au produit par le vendeur. Ce champ peut être omis.
softwareVersion	String	Version du logiciel. Ce champ doit être spécifié.
buildNumber	String	Numéro de mouture. Ce champ doit être spécifié.
buildDate	UTCTime	Date et heure de la mouture. Ce champ doit être spécifié.
issuedBy	String	URI de l'autorité de certification. Ce champ doit être spécifié.
issueDate	UTCTime	Spécifie l'heure à laquelle le *Certificat* a été émis par l'autorité de certification. Ce champ doit être spécifié.
vendorProductCertificate	ByteString	La forme codée DER du *Certificat* X.509 qui est affecté à l'application par le vendeur.
supportedProfiles []	structure	Liste des *Profiles* pris en charge.
organizationUri	String	Un URI qui identifie l'organisation qui a défini le profil.
profileID	String	Une chaîne qui identifie le *Profile.*
complianceTool	String	Une chaîne qui identifie l'outil ou la méthode de certification utilisé(e) pour les essais de conformité.
complianceDate	UTCTime	Date et heure de l'essai de conformité.
complianceLevel	enum ComplianceLevel	Une énumération qui spécifie le niveau de conformité du *Profile*. Cette énumération a les valeurs suivantes: UNTESTED_0 la fonction profilée n'a pas été soumise à essai avec succès. PARTIAL_1 la fonction profilée a été soumise partiellement à essai et a réussi aux essais critiques, tels que définis par l'autorité de certification. SELFTESTED_2 la fonction profilée a été soumise à essai avec succès en utilisant un système d'autotest autorisé par l'autorité de certification. CERTIFIED_3 la fonction profilée a été soumise à essai avec succès par un organisme d'essais autorisé par l'autorité de certification.
unsupportedUnitIds []	String	Les identificateurs pour les unités de conformité facultative qui n'ont pas été soumises à essai. Voir la CEI 62541-7 pour une explication détaillée.

7.33 Code d'état (StatusCode)

7.33.1 Généralités

Un *StatusCode* dans OPC UA est une valeur numérique qui est utilisée pour rapporter le résultat d'une opération accomplie par le *Serveur* OPC UA. Ce code peut avoir des informations de diagnostic associées qui décrivent le statut avec plus de détails; cependant, le code tout seul est censé fournir à des applications *Client* assez d'informations pour prendre des décisions quant à la manière de traiter les résultats d'un *Service* OPC UA.

Le *StatusCode* est un nombre entier non signé de 32 bits. Les 16 bits de poids fort représentent la valeur numérique du code qui doit être utilisé pour détecter des erreurs ou des conditions spécifiques. Les 16 bits de poids faible sont des fanions binaires qui contiennent des informations supplémentaires mais n'altèrent pas la signification du *StatusCode*.

Tous les *Clients* OPC UA doivent toujours vérifier le *StatusCode* associé à un résultat avant de l'utiliser. Les résultats auxquels sont associés un statut uncertain/warning (incertain/avertissement) doivent être utilisés avec prudence car ces résultats peuvent ne pas être valides pour toutes les situations. Les résultats avec un statut bad/failed (mauvais/défaillant) ne doivent jamais être utilisés.

Il convient que les *Serveur*s OPC UA retournent des *StatusCodes* good/success (Bon/succès) si l'opération s'est terminée normalement et le résultat est toujours valide. Des valeurs de *StatusCode* différentes peuvent fournir des informations supplémentaires au *Client*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Il convient que les *Serveurs* OPC UA utilisent *StatusCodes* uncertain/warning s'ils n'ont pas pu achever l'opération comme demandée par le *Client*, mais l'opération n'a toutefois pas complètement échoué.

Les affectations exactes des bits sont montrées au Tableau 163.

Tableau 163 – Affectations de bits de StatusCode

Champ	Plage de bits	Description
Severity	30:31	Indique si le *StatusCode* représente un état good (correct), bad (mauvais), uncertain (incertain). Ces bits ont les significations suivantes: Bonne exécution 00 Indique que l'opération c'est déroulée correctement et que le résultat associé peut être utilisé. Avertissement aléatoire 01 Indique que l'opération c'est en partie déroulée correctement et que le résultat associé peut dans certains cas ne pas être approprié. Echec sérieux 10 Indique que l'opération ne c'est pas déroulée correctement et que le résultat associé ne peut pas être utilisé. Réservé 11 Réservé pour des utilisations futures. Tous les clients traiteront le code status avec un niveau de sévérité "Mauvais".
Reserved	29:28	Réservé pour usage futur. Doit toujours être zéro.
SubCode	16:27	Le code est une valeur numérique affectée pour représenter différents états. Chaque code a un nom symbolique et une valeur numérique. Toutes les descriptions dans la spécification OPC UA se réfèrent au nom symbolique. La CEI 62541-6 fait correspondre les noms symboliques à une valeur numérique.
StructureChanged	15:15	Indique que la structure de la valeur de donnée associée a changé depuis la dernière *Notification*. Il convient que les *Clients* ne traitent pas la donnée sauf s'ils relisent les métadonnées. Les *Serveurs* doivent positionner ce bit si le *DataTypeEncoding* utilisé pour une *Variable* change. Le paragraphe 7.23 décrit comment le *DataTypeEncoding* est spécifié pour une *Variable*. Le bit est également mis si le type de donnée *Attribut* de la *Variable* change. Une *Variable* avec le type de donnée *BaseDataType* n'exige pas que le bit soit positionné lorsque le data type change. Les *Serveurs* doivent aussi positionner ce bit si l'*Attribut ArrayDimensions* ou *ValueRank* ou la *Property EnumStrings* des *DataType* de la *Variable* change. Ce bit est fourni pour avertir les *Clients* qui analysent des valeurs de données complexes que leurs programmes d'analyse pourraient faillir parce que la forme sérialisée de la valeur de donnée a changé. Ce bit n'a de sens que pour les *StatusCodes* retournés comme partie d'une *Notification* de changement de données ou de l'*HistoryRead*. Les *StatusCodes* utilisés dans d'autres contextes doivent toujours mettre ce bit à zéro.
SemanticsChanged	14:14	Indique que la sémantique de la valeur de donnée associée a changé. Il convient que les *Clients* traitent la valeur de donnée jusqu'à ce qu'ils relisent les métadonnées associées à la *Variable*. Il convient que les *Serveurs* positionnent ce bit si les métadonnées ont changé d'une manière susceptible de provoquer des erreurs d'application si le *Client* ne relit pas les métadonnées. Par exemple, un changement d'unités techniques pourrait créer des problèmes si le *Client* utilise la valeur pour effectuer des calculs. La CEI 62541-8 définit les conditions dans lesquelles un *Serveur* doit positionner ce bit pour une *Variable* DA. D'autres spécifications peuvent définir des conditions supplémentaires. Un *Serveur* peut définir d'autres conditions qui provoquent le positionnement de ce bit. Ce bit n'a de sens que pour les *StatusCodes* retournés comme partie d'une *Notification* de changement de données ou de l'*HistoryRead*. Les *StatusCodes* utilisés dans d'autres contextes doivent toujours mettre ce bit à zéro.
Reserved	12:13	Réservé pour des usages futurs. Doit toujours être zéro.
InfoType	10:11	Le type d'informations contenu dans les bits d'informations. Ces bits ont les significations suivantes: Pas utilisé 00 Les bits d'information ne sont pas utilisés et doivent être positionnés à zéro. Valeur de la donnée 01 Le StatusCode et ses bits d'information sont positionnés en fonction de la valeur de l'information retournée par le serveur. Réservé 1X Réservé pour des utilisations futures. Les bits d'information sont ignorés.
InfoBits	0:9	Bits d'informations supplémentaires qui qualifient le *StatusCode*. La structure de ces bits dépend du champ Info Type.

Le Tableau 164 décrit la structure des *InfoBits* lorsque l'Info Type est mis à *DataValue* (01).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 164 – DataValue InfoBits

<table>
<tr><th>Info Type</th><th>Plage de bits</th><th>DESCRIPTION</th></tr>
<tr><td>LimitBits</td><td>8:9</td><td>Les bits de limites associés à la valeur de donnée. Les bits de limites ont les significations suivantes:

Limit — Bits — Description
Aucune — 00 — La valeur peut être modifiée.
Faible — 01 — La valeur est à la limite inférieure pour la source de données.
Haute — 10 — La valeur est à la limite supérieure pour la source de données.
Constante — 11 — La valeur est constante et ne peut pas être modfiée.</td></tr>
<tr><td>Overflow</td><td>7</td><td>Si ce bit est positionné, les changements détectés n'ont pas tous été retournés depuis que le tampon de file d'attente du Serveur pour le MonitoredItem a atteint sa limite et a dû purger des données.</td></tr>
<tr><td>Reserved</td><td>5:6</td><td>Réservé pour des usages futurs. Doit toujours être zéro.</td></tr>
<tr><td>HistorianBits</td><td>0:4</td><td>Ces bits ne sont positionnés que lors de la lecture de données historiques. Ils indiquent la source de la valeur de donnée et fournissent des informations qui ont une incidence sur la manière dont le Client utilise la valeur de donnée. Les bits historiques ont la signification suivante:
Raw — XXX00 — Valeur de donnée brute.
Calculated — XXX01 — Valeur de donnée ayant fait l’objet d’un calcul.
Interpolated — XXX10 — Valeur de donnée interpolée.
Reserved — XXX11 — Non défini.
Partial — XX1XX — Valeur de donnée calculée avec intervalle incomplet.
Extra Data — X1XXX — Donnée brute cachant une autre donnée au niveau du même marqueur temporel.
Multi Value — 1XXXX — Des valeurs multiples correspondent au critère d’agrégat (c’est-à-dire des valeurs minimales multiples au niveau de différents marqueurs temporels dans le même intervalle).

La CEI 62541-11 décrit avec plus de détails comment ces bits sont utilisés.</td></tr>
</table>

7.33.2 StatusCodes communs

Le Tableau 165 définit les *StatusCodes* communs pour tous les résultats de *Service*. La CEI 62541-6 fait correspondre les noms symboliques à une valeur numérique.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 165 – Codes de résultats des services communs

Id symbolique	Description
Good	L'opération a réussi.
Good_CompletesAsynchronously	Le traitement sera accompli de manière asynchrone.
Good_SubscriptionTransferred	L'abonnement a été transféré à une autre session.
Bad_CertificateHostNameInvalid	Le *HostName* utilisé pour se connecter à un *Serveur* ne correspond pas au *HostName* contenu dans le Certificat.
Bad_CertificateIssuerRevocationUnknown	Il n'était pas possible de déterminer si le Certificat de l'émetteur a été annulé.
Bad_CertificateIssuerUseNotAllowed	Le *Certificat* de l'émetteur ne peut pas être utilisé pour l'opération demandée.
Bad_CertificateIssuerTimeInvalid	Le *Certificat* d'un émetteur a expiré ou n'est pas encore valide.
Bad_CertificateIssuerRevoked	Le *Certificat* d'émetteur a été annulé.
Bad_CertificateInvalid	Le certificat fourni comme paramètre n'est pas valide.
Bad_CertificateRevocationUnknown	Il n'était pas possible de déterminer si le *Certificat* a été annulé.
Bad_CertificateRevoked	Le *Certificat* a été annulé.
Bad_CertificateTimeInvalid	Le *Certificat* a expiré ou n'est pas encore valide.
Bad_CertificateUriInvalid	L'URI spécifié dans l'*ApplicationDescription* ne correspond pas à l'URI contenu dans le *Certificat*.
Bad_CertificateUntrusted	Le *Certificat* n'est pas fiable.
Bad_CertificateUseNotAllowed	Le *Certificat* ne peut pas être utilisé pour l'opération demandée.
Bad_CommunicationError	Une erreur de communication de bas niveau s'est produite.
Bad_DataTypeIdUnknown	L'objet d'extension ne peut pas être (dé)sérialisé car l'identificateur de type de donnée n'est pas reconnu.
Bad_DecodingError	Décodage arrêté en raison d'une donnée non valide dans le flux.
Bad_EncodingError	Codage arrêté en raison d'une donnée non valide dans les objets sérialisés.
Bad_EncodingLimitsExceeded	Les limites de codage/décodage des messages imposées par la *Pile de Communication* ont été dépassées.
Bad_IdentityTokenInvalid	Le jeton d'identité d'utilisateur n'est pas valide.
Bad_IdentityTokenRejected	Le jeton d'identité d'utilisateur est valide mais le serveur l'a rejeté.
Bad_InternalError	Une erreur interne s'est produite à la suite d'une erreur de programmation ou de configuration.
Bad_InvalidArgument	Un ou plusieurs arguments ne sont pas valides. Chaque service définit des *StatusCodes* spécifiques au paramètre et ces *StatusCodes* doivent être utilisés au lieu de ce code d'erreur générale. Ce code d'erreur doit être uniquement utilisé par la *Pile de Communication* et dans des services où il est défini dans la liste de *StatusCodes* valides pour le service.
Bad_InvalidState	L'opération ne pouvait pas être menée à bien car l'objet est fermé, non initialisé ou dans quelque autre état non valide.
Bad_InvalidTimestamp	L'horodatage est hors la plage permise par le serveur.
Bad_NonceInvalid	Le nonce s'avère être une valeur non aléatoire ou il n'a pas la longueur correcte.
Bad_NothingToDo	Il n'y avait rien à faire car le client a passé une liste d'opérations sans éléments.
Bad_OutOfMemory	Mémoire insuffisante pour parachever l'opération.
Bad_RequestCancelledByClient	La demande était annulée par le client.
Bad_RequestTooLarge	La taille du message de demande dépasse les limites fixées par le serveur.
Bad_ResponseTooLarge	La taille du message de réponse dépasse les limites fixées par le client.
Bad_RequestHeaderInvalid	L'en-tête pour la demande est absent ou n'est pas valide.
Bad_ResourceUnavailable	Une ressource du système d'exploitation n'est pas disponible.
Bad_SecureChannelIdInvalid	La voie sécurisée spécifiée n'est plus valide.
Bad_SecurityChecksFailed	Une erreur s'est produite lors de la vérification de la sécurité.
Bad_ServerHalted	Le serveur s'est arrêté et ne peut traiter aucune demande.
Bad_ServerNotConnected	L'opération ne pouvait pas être achevée car le client n'est pas connecté au serveur.
Bad_ServerUriInvalid	L'URI de *Serveur* n'est pas valide.
Bad_ServiceUnsupported	Le serveur ne prend pas en charge le service demandé.
Bad_SessionIdInvalid	L'identificateur de session n'est pas valide.
Bad_SessionClosed	La session était fermée par le client.
Bad_SessionNotActivated	La session ne pouvait pas être utilisée car ActivateSession n'a pas été appelé.
Bad_Shutdown	L'opération était annulée car l'application se fermait.
Bad_SubscriptionIdInvalid	L'identificateur d'abonnement n'est pas valide.
Bad_Timeout	L'opération s'est interrompue par temporisation.
Bad_TimestampsToReturnInvalid	Le paramètre marqueurs temporels à retourner n'est pas valide.
Bad_TooManyOperations	La requête ne pouvait pas être traitée car elle spécifiait de trop nombreuses opérations.
Bad_UnexpectedError	Une erreur inattendue s'est produite.
Bad_UnknownResponse	Une réponse non reconnue était reçu du serveur.
Bad_UserAccessDenied	L'utilisateur n'a pas la permission d'effectuer l'opération demandée.
Bad_ViewIdUnknown	L'identificateur de vue ne renvoie pas à un Nœud de vue valide.
Bad_ViewTimestampInvalid	Le marqueur temporel de la vue n'est pas disponible ou n'est pas pris en charge.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Id symbolique	Description
Bad_ViewParameterMismatchInvalid	Les paramètres de vue ne sont pas cohérents les uns avec les autres.
Bad_ViewVersionInvalid	La version de la vue n'est pas disponible ou n'est pas prise en charge.

Le Tableau 166 définit les *StatusCodes* communs pour tous les résultats de niveau opération. La CEI 62541-6 fait correspondre les noms symboliques à une valeur numérique. Les codes de résultats de *Services* communs peuvent aussi être contenus dans le niveau opération.

Tableau 166 – Codes de résultats de niveau opération communs

Id symbolique	Description
Good_Clamped	La valeur écrite a étét acceptée mais a été bloquée.
Good_Overload	L'échantillonnage a ralenti en raison de limitations de ressources.
Uncertain	La valeur est incertaine mais aucune cause spécifique n'est connue.
Bad	La valeur est bad (mauvaise) mais aucune raison spécifique n'est connue.
Bad_AttributeIdInvalid	L'attribut n'est pas pris en charge pour le nœud spécifié.
Bad_BrowseDirectionInvalid	Le sens du parcours dans la navigation n'est pas valide.
Bad_BrowseNameInvalid	Le nom de navigation n'est pas valide.
Bad_ContentFilterInvalid	Le filtre de contenu n'est pas valide.
Bad_ContinuationPointInvalid	Le point de continuation fourni n'est plus valide.
Bad_DataEncodingInvalid	Le codage de données n'est pas valide.
Bad_DataEncodingUnsupported	Le serveur ne prend pas en charge le codage de données demandé pour le nœud.
Bad_EventFilterInvalid	Le filtre d'événements n'est pas valide.
Bad_FilterNotAllowed	Un filtre de surveillance ne peut pas être utilisé en combinaison avec l'attribut spécifié.
Bad_FilterOperandInvalid	L'opérande utilisé dans le filtre de contenu n'est pas valide.
Bad_HistoryOperationInvalid	Le paramètre détails d'historique n'est pas valide.
Bad_HistoryOperationUnsupported	Le serveur ne prend pas en charge l'opération demandée.
Bad_IndexRangeInvalid	La syntaxe du paramètre plage d'indices n'est pas valide
Bad_IndexRangeNoData	Il n'existe pas de données dans la plage d'indices spécifiée.
Bad_MonitoredItemFilterInvalid	La paramètre filtre d'éléments surveillés n'est pas valide.
Bad_MonitoredItemFilterUnsupported	Le serveur ne prend pas en charge le filtre d'éléments surveillés demandé.
Bad_MonitoredItemIdInvalid	L'identificateur d'élément surveillé ne renvoie pas à un élément surveillé valide.
Bad_MonitoringModeInvalid	Le mode de surveillance n'est pas valide.
Bad_NoCommunication	La communication avec la source de données est définie mais n'a pas été établie et aucune dernière valeur connue n'est disponible. Cet état/sous-état est utilisé pour les valeurs placées en cache avant que la première valeur soit reçue.
Bad_NoContinuationPoints	L'opération ne pouvait pas être traitée car tous les points de continuation ont été attribués.
Bad_NodeClassInvalid	La classe de nœuds n'est pas valide.
Bad_NodeIdInvalid	La syntaxe de l'identificateur de nœud n'est pas valide.
Bad_NodeIdUnknown	L'identificateur de nœud renvoie à un nœud qui n'existe pas dans l'Espace d'Adresses du serveur.
Bad_NoDeleteRights	Le *Serveur* ne permettra pas la suppression du nœud.
Bad_NodeNotInView	Le nodeToBrowse (nœud à parcourir) ne fait pas partie de la vue.
Bad_NotFound	Un élément demandé n'a pas été trouvé ou l'opération de recherche s'est terminée sans succès.
Bad_NotImplemented	L'opération demandée n'est pas implémentée.
Bad_NotReadable	Le niveau d'accès ne permet pas la lecture ou l'abonnement au Noeud.
Bad_NotSupported	L'opération demandée n'est pas prise en charge.
Bad_NotWritable	Le niveau d'accès ne permet pas l'écriture sur le Noeud.
Bad_ObjectDeleted	L'objet ne peut pas être utilisé car il a été supprimé.
Bad_OutOfRange	La valeur était hors plage.
Bad_ReferenceTypeIdInvalid	L'identificateur de type "référence" ne renvoie pas à un nœud de type "référence" valide.
Bad_SourceNodeIdInvalid	L'identificateur de nœud source ne renvoie pas à un nœud valide.
Bad_StructureMissing	Un paramètre structuré obligatoire était absent ou null.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Id symbolique	Description
Bad_TargetNodeIdInvalid	L'identificateur de nœud cible ne renvoie pas à un nœud valide.
Bad_TypeDefinitionInvalid	L'identificateur de nœud de définition de type ne référence pas un nœud de type approprié.
Bad_TypeMismatch	La valeur fournie pour l'attribut n'est pas du même type que la valeur de l'attribut.
Bad_WaitingForInitialData	En attente que le serveur obtienne des valeurs de la part de la source de données sous-jacente. Après avoir créé un *MonitoredItem*, il peut prendre un certain temps au serveur pour obtenir effectivement des valeurs pour ces éléments. Dans ces cas, le serveur peut facultativement envoyer une *Notification* avec ce statut avant la *Notification* avec la première valeur valide.

7.34 Marqueurs temporel à retourner (TimestampsToReturn)

Le *TimestampsToReturn* est une énumération qui spécifie les *Attributs Timestamp* à transmettre pour des *MonitoredItems* ou des Noeuds dans *HistoryRead*. Les valeurs de ce paramètre sont définies au Tableau 167.

Tableau 167 – Valeurs de TimestampsToReturn

Valeur	Description
SOURCE_0	Retourner le marqueur temporel de la source. S'il est utilisé dans *HistoryRead*, le marqueur temporel de la source est utilisé pour déterminer les valeurs de données historiques qui seront retournées.
SERVER_1	Retourner le marqueur temporel du *Serveur*. S'il est utilisé dans *HistoryRead*, le marqueur temporel du *Serveur* est utilisé pour déterminer les valeurs de données historiques qui seront retournées.
BOTH_2	Retourner les marqueurs temporels tant de la source que du *Serveur*. S'il est utilisé dans *HistoryRead*, le marqueur temporel de la source est utilisé pour déterminer les valeurs de données historiques qui seront retournées.
NEITHER_3	Ne retourner aucun des marqueurs temporels. Il s'agit de la valeur par défaut pour les *MonitoredItems* si une valeur *Variable* n'est pas actuellement accessible. Pour *HistoryRead*, il ne s'agit pas d'un paramétrage valide.

7.35 Paramètres du jeton identifiant l'utilisateur (UserIdentityToken)

7.35.1 Vue générale

La structure *UserIdentityToken* utilisée dans le *Server Service Set* (Jeu de services serveur) permet aux Clients de spécifier l'identité de l'utilisateur pour le compte duquel ils agissent. Le mécanisme exact utilisé pour identifier des utilisateurs dépend de la configuration du système. Les différents types de jetons d'identité reposent sur les mécanismes les plus communs qui sont utilisés dans les systèmes à l'heure actuelle. Le Tableau 168 définit le jeu courant de jetons d'identité d'utilisateur. Le type *ExtensibleParamter* est défini en 7.11.

Tableau 168 – TypeIds du paramètre UserIdentityToken

Id symbolique	Description
AnonymousIdentityToken	Aucune information d'utilisateur n'est disponible.
UserNameIdentityToken	Un utilisateur identifié par le nom utilisateur et le mot de passe.
X509IdentityToken	Un utilisateur identifié par un *Certificat* X509v3.
IssuedIdentityToken	Un utilisateur identifié par un WS-*SecurityToken*.

Le Client doit toujours prouver la possession d'un *UserIdentityToken* lorsqu'il le passe au *Serveur*. Certains jetons incluent un secret tel qu'un mot de passe que le *Serveur* acceptera comme preuve. Afin de protéger ces secrets, le *Token* doit être chiffré avant qu'il ne soit passé au *Serveur*. D'autres types de jetons permettent au Client de créer une signature avec le secret associé au *Token*. Dans ces cas, le Client prouve la possession d'un *UserIdentityToken* en accolant le dernier *ServerNonce* au *ServerCertificate* et utilise le secret pour produire une *Signature* qui est passée au *Serveur*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Chaque *UserIdentityToken* autorisé par un *Endpoint* doit avoir une *UserTokenPolicy* spécifiée dans l'*EndpointDescription*. Le *UserTokenPolicy* spécifie la *SecurityPolicy* qui est à utiliser pour le chiffrement ou la signature. Si cette *SecurityPolicy* est omise, le Client utilise la *SecurityPolicy* figurant dans l'*EndpointDescription*. Si la *SecurityPolicy* concordante est mise None, il n'est pas requis de chiffrement ou de signature. Il est recommandé que les Applications ne mettent jamais la SecurityPolicy à None pour des UserTokens qui incluent un secret car ces secrets pourraient être utilisés par un attaquant pour avoir accès au système.

Le Tableau 169 décrit comment sérialiser des *UserIdentityTokens* avant d'appliquer le chiffrement.

Tableau 169 – Format de jeton chiffré UserIdentityToken

Nom	Type	Description
length	Byte[4]	La longueur des données chiffrées, y compris le *ServerNonce* mais non compris le champ *length*. Ce champ est un nombre entier non signé de quatre octets où les octets de poids faible apparaissent en premier.
tokenData	Byte[*]	La valeur de jeton.
serverNonce	Byte[*]	Le dernier *ServerNonce* retourné par le serveur dans la réponse *CreateSession* ou *ActivateSession*.

7.35.2 AnonymousIdentityToken

L'*AnonymousIdentityToken* est utilisé pour indiquer que le Client n'a pas d'authentifiants d'utilisateur.

Le Tableau 170 définit le paramètre AnonymousIdentityToken.

Tableau 170 – AnonymousIdentityToken

Nom	Type	Description
AnonymousIdentityToken	structure	Une identité d'utilisateur anonyme.
policyId	String	Un identificateur pour la *UserTokenPolicy* à laquelle le jeton se conforme. La structure *UserTokenPolicy* est définie en 7.36.

7.35.3 UserNameIdentityToken

Le *UserNameIdentityToken* est utilisé pour passer de simples authentifiants de nom d'utilisateur/mot de passe au *Serveur*.

Ce jeton doit être chiffré si la *SecurityPolicy* l'exige. Il convient que le *Serveur* spécifie une *SecurityPolicy* pour la *UserTokenPolicy* si le *Canal Sécurisé* à une *SecurityPolicy* None.

Si le jeton est chiffré, le mot de passe doit être converti en *ByteString* UTF8 puis sérialisé comme montré au Tableau 169.

Le *Serveur* doit déchiffrer le mot de passe et vérifier le *ServerNonce*.

Si la SecurityPolicy est None, le mot de passe contient seulement le mode de passe codé UTF-8.

Le Tableau 171 définit le paramètre *UserNameIdentityToken*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 171 – UserNameIdentityToken

Nom	Type	Description
UserNameIdentityToken	structure	UserName value.
policyId	String	Un identificateur pour la *UserTokenPolicy* à laquelle le jeton se conforme. La structure *UserTokenPolicy* est définie en 7.36.
userName	String	Une chaîne qui identifie l'utilisateur.
password	ByteString	Le mot de passe pour l'utilisateur. Ce paramètre doit être chiffré avec le Certificat du *Serveur* à l'aide de l'algorithme spécifié par la *SecurityPolicy*.
encryptionAlgorithm	String	Une chaîne contenant l'URI de l'*encryptionAlgorithm*. Les valeurs de la chaîne URI sont des noms définis qui peuvent être utilisés comme partie des profils de sécurité spécifiés dans la CEI 62541-7. Ce paramètre est null si le mot de passe n'est pas chiffré.

7.35.4 X509IdentityTokens

Le *X509IdentiyToken* est utilisé pour passer un *Certificat* X509v3 qui est émis par l'utilisateur.

Ce jeton doit toujours être accompagné par une signature si la *SecurityPolicy* l'exige. Il convient que le *Serveur* spécifie une *SecurityPolicy* pour la *UserTokenPolicy* si le *Canal Sécurisé* à une *SecurityPolicy* None.

Le Tableau 172 définit le paramètre X509IdentityToken.

Tableau 172 – Jeton d'identité X509

Nom	Type	Description
X509IdentityToken	structure	Valeur X509v3.
policyId	String	Un identificateur pour la *UserTokenPolicy* à laquelle le jeton se conforme. La structure *UserTokenPolicy* est définie en 7.36.
certificateData	ByteString	Le *Certificat* X509 au format DER.

7.35.5 IssuedIdentityToken

L'*IssuedIdentityToken* est utilisé pour passer des *SecurityToken*s conformes à WS-Security au *Serveur*.

WS-Security définit un certain nombre de profils de jetons qui peuvent être utilisés pour représenter différents types de *SecurityTokens*. Par exemple, les jetons Kerberos et SAML ont des profils de jetons WSS et doivent être échangés dans OPC UA comme des jetons de sécurité XML.

Il convient que les jetons WSS X509 et UserName ne soient pas échangés comme des jetons de sécurité XML. Il convient que les applications OPC UA utilisent les jetons d'identité OPC UA pour passer les informations contenues dans ces types de *SecurityTokens* WSS.

Ces jetons peuvent être chiffrés ou exiger une signature. La CEI 62541-7 définit des profils qui incluent la sécurité relative à l'utilisateur et qui incluent également toutes les exigences éventuelles relatives au chiffrement des signatures lorsque la signature est requise. Des profils de sécurité supplémentaires spécifient des algorithmes de chiffrement et de signature.

Si le jeton est chiffré, l'expression en XML doit être convertie en *ByteString* UTF8 puis sérialisé comme montré au Tableau 169.

Le *Serveur* doit déchiffrer le tokenData et vérifier le *ServerNonce*.

Si la SecurityPolicy est None ou si le jeton exige seulement une signature, le tokenData contient la représentation en XML codé UTF-8 du jeton.

Le Tableau 173 définit le paramètre *IssuedIdentityToken*.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 173 – IssuedIdentityToken

Nom	Type	Description
IssuedIdentityToken	structure	Valeur WSS.
policyId	String	Un identificateur pour la *UserTokenPolicy* à laquelle le jeton se conforme. La structure *UserTokenPolicy* est définie en 7.36.
tokenData	ByteString	La représentation en XML du jeton. Ce paramètre peut être chiffré avec le Certificat du *Serveur*.
encryptionAlgorithm	String	Une chaîne contenant l'URI de l'*encryptionAlgorithm*. La liste des noms définis par l'OPC UA qui peuvent être utilisés est spécifiée dans la CEI 62541-7. Ce paramètre est null si le jeton n'est pas chiffré.

7.36 Politique pour le jeton utilisateur (UserTokenPolicy)

Les composantes de ce paramètre sont définies au Tableau 174.

Tableau 174 – UserTokenPolicy

Nom	Type	Description
UserTokenPolicy	structure	Spécifie un *UserIdentityToken* qu'un *Serveur* acceptera.
policyId	String	Un identificateur pour la UserTokenPolicy affecté par le *Serveur*. Le Client spécifie cette valeur lorsqu'il construit un UserIdentityToken qui se conforme à la politique. Cette valeur est seulement unique dans le contexte d'un *Serveur* seul.
tokenType	Enum UserIdentity TokenType	Le type de jeton d'identité d'utilisateur requis. Cette valeur est une énumération avec les valeurs suivantes: ANONYMOUS_0 Aucun jeton n'est requis. USERNAME_1 Un jeton de nom d'utilisateur/de mot de passe. CERTIFICATE_2 Un jeton de certificat X509v3. ISSUEDTOKEN_3 N'importe quel jeton défini par WS-Security. Un tokenType ANONYMOUS indique que le *Serveur* n'exige pas d'identification d'utilisateur. Dans ce cas, le *certificate* d'instance d'application *Client* est utilisé comme identification d'utilisateur.
issuedTokenType	String	Un URI pour le type de jeton. La CEI 62541-7 définit les URI pour les types communs de jetons émis. Les vendeurs peuvent spécifier leur propre jeton. Ce champ peut seulement être spécifié si *TokenType* est ISSUEDTOKEN. Les jetons WS-Security sont parfois identifiés par des XML QualifiedNames. Un URI pour le jeton peut être construit en accolant le nom à l'espace de noms avec un séparateur ':'. Le XML QualifiedName peut être reconstruit en recherchant le dernier délimiteur ':'.
issuerEndpointUrl	String	Un URL facultatif pour le service émettant le jeton. La signification de cette valeur dépend de l'*issuedTokenType.*
securityPolicyUri	String	La politique de sécurité à appliquer pour chiffrer ou signer le *UserToken* lorsqu'il est passé au *Serveur* dans la demande *ActivateSession*. Le paragraphe 7.35 décrit comment ce paramètre est utilisé. La politique de sécurité pour le Canal Sécurisé est appliquée si cette valeur est omise.

7.37 Description d'une vue (ViewDescription)

Les composantes de ce paramètre sont définies au Tableau 175.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau 175 – ViewDescription

Nom	Type	Description
ViewDescription	structure	Spécifie une Vue.
viewId	NodeId	*NodeId* de la Vue à interroger (*Query*). Une valeur null indique l'Espace d'Adresses tout entier.
timestamp	UTCTime	La date et l'heure souhaitées. La version correspondante est celle qui a l'horodatage de création précédente le plus proche. Le paramètre *Timestamp* ou bien le paramètre *viewVersion* peut être établi par un *Client*, mais pas les deux. Si *ViewVersion* est positionné, ce paramètre doit être null.
viewVersion	UInt32	Le numéro de version pour la Vue souhaitée. Lorsque des Noeuds sont ajoutés à une Vue ou en sont retirés, la valeur de *ViewVersion Property* d'une Vue est mise à jour. Le paramètre *Timestamp* ou bien le paramètre *viewVersion* peut être établi par un *Client*, mais pas les deux. La *Property* ViewVersion est définie dans la CEI 62541-3. Si *timestamp* est mis, ce paramètre doit être 0. La vue courante est utilisée si timestamp est null et viewVersion est 0.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe A
(informative)

Définitions BNF

A.1 Vue d'ensemble sur BNF

La notation BNF (Backus-Naur form) utilisée dans cette Annexe utilise `<´ et `>´ pour marquer des symboles, `[´ et `]´ pour identifier des chemins facultatifs et `|´ pour identifier des alternatives. Si les symboles '(' et ')' sont utilisés, ils indiquent des ensembles.

A.2 BNF de RelativePath

Un *RelativePath* est une structure qui décrit une séquence de *Références* et de Noeuds à suivre. La présente Annexe décrit un format de texte pour un RelativePath qui peut être utilisé dans une documentation ou dans des fichiers servant à stocker des informations de configuration.

Les composantes d'un format de texte pour *RelativePath* sont spécifiées au Tableau A.1.

Tableau A.1 – RelativePath

Symbole	Signification
/	Le caractère slash (barre oblique) indique que le *Serveur* doit suivre tout sous-type de *HierarchicalReferences*.
.	Le caractère point final (point) indique que le *Serveur* doit suivre tout sous-type d'un *Aggregates ReferenceType* (type de référence agrégats).
<[#!ns:]ReferenceType>	Une chaîne délimitée par les symboles «<» et «>» spécifie le *BrowseName* d'un *ReferenceType* à suivre. Par défaut, n'importe quelles *Références* des sous-types de *ReferenceType* sont suivies également. Un «#» placé devant le BrowseName indique qu'il convient de ne pas suivre les sous-types. Un «!» devant le BrowseName sert à indiquer qu'il convient de suivre la *Référence* inverse. Le *BrowseName* peut être qualifié par un indice d'espace de noms (indiqué par un préfixe numérique suivi de deux points). Cet indice d'espace de noms est utilisé pour spécifier la composante espace de noms du *BrowseName* pour le *ReferenceType*. Si le préfixe d'espace de noms est omis, l'indice d'espace de noms 0 est utilisé.
[ns:]BrowseName	Une chaîne qui suit un symbole «/», «.» ou «>» spécifie le *BrowseName* d'un Noeud à retourner ou à suivre. Ce BrowseName peut être préfixé par son indice d'espace de noms. Si le préfixe d'espace de noms est omis, l'indice d'espace de noms 0 est utilisé. L'omission du *BrowseName* final d'un chemin équivaut à une opération joker qui correspond à tous les Noeuds qui sont la cible de la *Référence* spécifiée par le chemin.
&	Le signe & est le caractère d'échappement. Il sert à spécifier des caractères réservés qui apparaissent au sein d'un *BrowseName*. Un caractère réservé est échappé en insérant le '&' devant lui. Des exemples de *BrowseNames* avec des caractères échappés sont: Nom de chemin de navigation reçu — Se résout en "&/Name_1" — "/Name_1" "&.Name_2" — ".Name_2" "&:Name_3" — ":Name_3" "&&Name_4" — "&Name_4"

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le Tableau A.2 donne des exemples de *RelativePaths* spécifiés avec le format de texte.

Tableau A.2 – Exemples de RelativePath

Chemin de navigation	Description
"/2:Block&.Output"	Suit toute *Référence* hiérarchique directe avec la cible *BrowseName* = "2:Block.Output".
"/3:Truck.0:NodeVersion"	Suit toute *Référence* hiérarchique directe avec la cible *BrowseName* = "3:Truck" et de là une *Aggregates Reference* directe jusqu'à la cible avec *BrowseName* "0:NodeVersion".
"<1:ConnectedTo>1:Boiler/1:HeatSensor"	Suit toute Référence directe avec un *BrowseName* = '1:ConnectedTo' et trouve les cibles avec *BrowseName* = '1:Boiler'. De là, il suit toute *Référence* hiérarchique et trouve des cibles avec *BrowseName* = '1:HeatSensor'.
"<1:ConnectedTo>1:Boiler/"	Suit toute Référence directe avec un *BrowseName* = '1:ConnectedTo' et trouve les cibles avec *BrowseName* = '1:Boiler'. De là, il trouve toutes les cibles de *Références* hiérarchiques.
"<0:HasChild>2:Wheel"	Suit toute Référence directe avec *BrowseName* = 'HasChild' et qualifié par l'espace de noms par défaut de l'OPC UA. Puis il trouve les cibles avec *BrowseName* = 'Wheel' qualifiées avec l'indice d'espace de noms '2'.
"<!HasChild>Truck"	Suit toute Référence inverse avec un *BrowseName* = 'HasChild' (à savoir, il suit la *Référence HasParent*). Puis il trouve les cibles avec *BrowseName* = 'Truck'. Dans les deux cas, la composante espace de noms du *BrowseName* est supposée être 0.
"<0:HasChild>"	Trouve toutes les cibles des *Références* directes avec un *BrowseName* = 'HasChild' et qualifié avec l'espace de noms par défaut de l'OPC UA.

La notation BNF ci-après décrit la syntaxe du format de texte de *RelativePath*.

```
<relative-path>  ::= <reference-type> <browse-name> [relative-path]

<reference-type> ::= '/' | '.' | '<' ['#'] ['!'] <browse-name> '>'

<browse-name>    ::= [<namespace-index> ':'] <name>

<namespace-index>      ::= <digit> [<digit>]

<digit>          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |
'8' | '9'

<name>
      ::= (<name-char> | '&' <reserved-char>) [<name>]

<reserved-char>  ::= '/' | '.' | '<' | '>' | ':' | '#' | '!' | '&'

<name-char>      ::=  Tous les caractères valides pour une chaîne String (voir la
            CEI 62541-3) en excluant les reserved-chars (caractères réservés).
```

A.3 BNF de NumericRange

La notation BNF ci-après décrit la syntaxe du type de paramètre NumericRange.

```
<numeric-range>  ::= <dimension> [',' <dimension>]

<dimension>      ::= <index> [':' <index>]

<index>          ::= <digit> [<digit>]

<digit>          ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' |
'8' | '9'
```

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe B
(informative)

Composition des Filtres et exemples d'interrogation (Query)

B.1 Exemple simples de ContentFilter

B.1.1 Vue générale

Ces exemples présentent des compostions de filtres très simples. Un filtre semblable à ces exemples peut être utilisé pour traiter des événements.

Les conventions suivantes s'appliquent à ces exemples et traite de la manière d'utiliser les opérandes des attributs (pour une définition de cet opérande, voir 7.4.4):

- AttributeOperand: Se réfère à un Noeud, à l'*Attribut* d'un Noeud ou à l'*Attribut Value* d'une *Property* associée à un Noeud. Dans les exemples, des noms en caractères de NodeIds sont utilisés en lieu et place d'un nodeId réel; cela s'applique aussi aux identificateurs d'attributs (Attribute Ids).
- La représentation des chemins relatifs sous forme de chaînes est utilisée en lieu et place de la structure réelle.
- Le NamespaceIndex utilisé dans tous les exemples est 12 (tout aussi bien il aurait pu être 4 ou 23 ou n'importe quelle valeur). Pour plus d'informations relatives à NamespaceIndex, voir la CEI 62541-3. L'utilisation du NamespaceIndex montre que le modèle d'informations utilisé dans les exemples n'est pas un modèle défini par la présente norme, mais qu'il est un modèle créé pour les exemples.

B.1.2 Exemple 1

Par exemple, la logique décrite par «(((AType.A = 5) or InList(BType.B, 3,5,7)) and BaseObjectType.displayName LIKE “Main%”)» se traduirait par un arbre logique tel que présenté à la Figure B.1 et un ContentFilter tel que présenté au Tableau B.1. Pour que cet exemple retourne quoi que ce soit, AType et BType doivent tous deux être des sous-types de BaseObjectType, ou bien l'opération “And” qui en résulte serait toujours fausse.

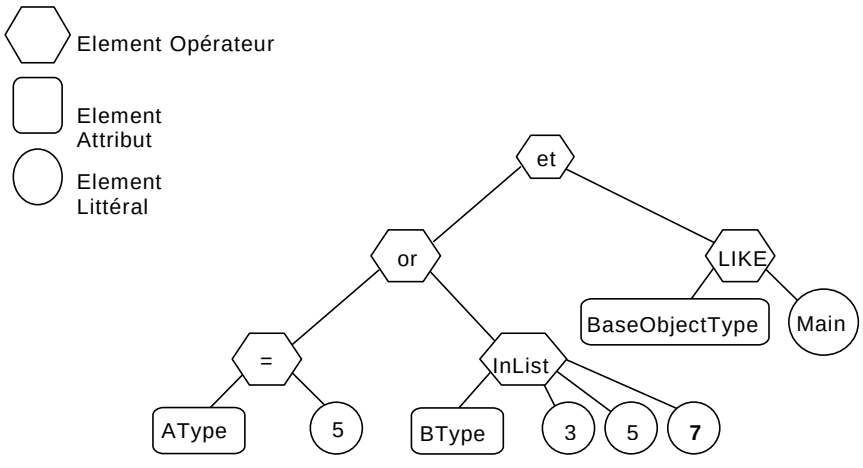


Figure B.1 – Exemple d'arbre logique de filtre

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le Tableau B.1 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.1 – Exemple de ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	et	ElementOperand = 1	ElementOperand = 4		
1	Or	ElementOperand = 2	ElementOperand = 3		
2	Equals	AttributeOperand = NodeId: AType, BrowsePath: ".12:A", AttributeId:value	LiteralOperand = '5'		
3	InList	AttributeOperand = NodeId: BType, BrowsePath: ".12:B", AttributeId:value	LiteralOperand = '3'	LiteralOperand = '5'	LiteralOperand = '7'
4	Like	AttributeOperand = NodeId: BaseObjectType, BrowsePath: ".", AttributeId: displayName	LiteralOperand = "Main%"		

B.1.3 Exemple 2

A titre d'autre exemple, un filtre pour sélectionner tous les *SystemEvents* (y compris les types dérivés) qui sont contenus dans la *View* Area1 ou la *View* Area2 se traduirait par un arbre logique tel que montré à la Figure B.2 et par un ContentFilter tel que montré au Tableau B.2.

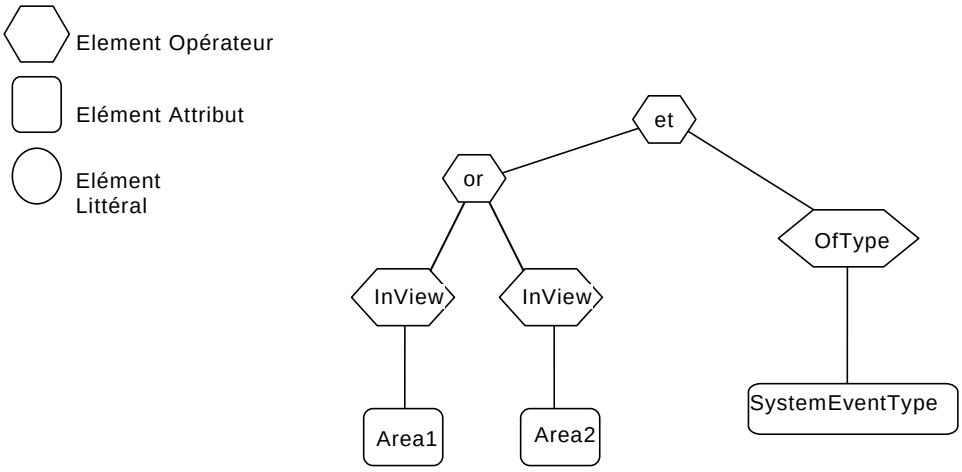


Figure B.2 – Exemple d'arbre logique de filtre

Le Tableau B.2 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.2 – Exemple de ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]
0	et	ElementOperand = 1	ElementOperand = 4
1	Or	ElementOperand = 2	ElementOperand = 3
2	InView	AttributeOperand = NodeId: Area1, BrowsePath: ".", AttributeId: NodeId	
3	InView	AttributeOperand = NodeId: Area2, BrowsePath: ".", AttributeId: NodeId	
4	OfType	AttributeOperand = NodeId: SystemEventType, BrowsePath: ".", AttributeId: NodeId"	

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.2 Exemples complexes de ContentFilters (Interrogations)

B.2.1 Vue générale

Ces exemples d'interrogation illustrent des *ContentFilters complexes.* Les conventions suivantes s'appliquent à ces exemples en ce qui concerne la manière d'utiliser les opérandes d'attribut (pour une définition de ces opérandes, voir 7.4.4):

- AttributeOperand: Se réfère à un Noeud, à l'*Attribut* d'un Noeud ou à l'*Attribut Value* d'une *Propriété* associée à un Noeud. Dans les exemples, des noms en caractères de NodeIds sont utilisés en lieu et place d'un nodeId réel; cela s'applique aussi aux identificateurs d'attributs (Attribute Ids).
- La représentation des chemins relatifs sous forme de chaînes est utilisée en lieu et place de la structure réelle.
- Le NamespaceIndex utilisé dans tous les exemples est 12 (il aurait pu être tout aussi bien 4 ou 23 ou n'importe quelle valeur). Pour plus d'informations relatives à NameSpacesIndex, voir la CEI 62541-3. L'utilisation du NamespaceIndex montre que le modèle d'informations utilisé dans les exemples n'est pas un modèle défini par cette norme, mais que c'est un modèle créé pour les exemples.

B.2.2 Modèle de type utilisé

Les exemples suivants utilisent le modèle de type décrit ci-dessous.

Nouveaux types de références:
"HasChild" dérivé de HierarchicalReference.
"HasAnimal" dérivé de HierarchicalReference.
"HasPet" dérivé de HasAnimal.
"HasFarmAnimal" dérivé de HasAnimal.
"HasSchedule" dérivé de HierarchicalReference.

PersonType dérivé de BaseObjectType ajoute:
HasProperty "LastName".
HasProperty "FirstName".
HasProperty "StreetAddress".
HasProperty "City".
HasProperty "ZipCode".
Peut avoir une référence HasChild à un nœud de type PersonType.
Peut avoir une référence HasAnimal à un nœud de type AnimalType (ou sous-type de ce type *Reference*).

AnimalType dérivé de BaseObjectType ajoute:
Peut avoir une référence HasSchedule à un nœud de type FeedingScheduleType.
HasProperty "Name".

DogType dérivé de AnimalType ajoute:
HasProperty "NickName".
HasProperty "DogBreed".
HasProperty "License".

CatType dérivé de AnimalType ajoute:
HasProperty "NickName".
HasProperty "CatBreed".

PigType dérivé de AnimalType ajoute:
HasProperty "PigBreed".

ScheduleType dérivé de BaseObjectType ajoute:
HasProperty "Period".

FeedingScheduleType dérivé de ScheduleType ajoute:

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

HasProperty "Food".
HasProperty "Amount".

AreaType dérivé de *BaseObjectType* est juste un simple *Folder* et ne contient pas de *Properties*.

Cet exemple de système de type est montré à la Figure B.3. Dans cette Figure, la notation OPC UA est utilisée pour toutes les Références à des types d'*Objet*, *Propriétés* et sous-types. En outre, les *Références* prises en charge sont contenues dans une boîte interne. Les références réelles existent uniquement dans les instances et, de ce fait, la Figure ne montre pas de connexions aux autres *Objets* et elles peuvent être des sous-types de la *Référence* répertoriée.

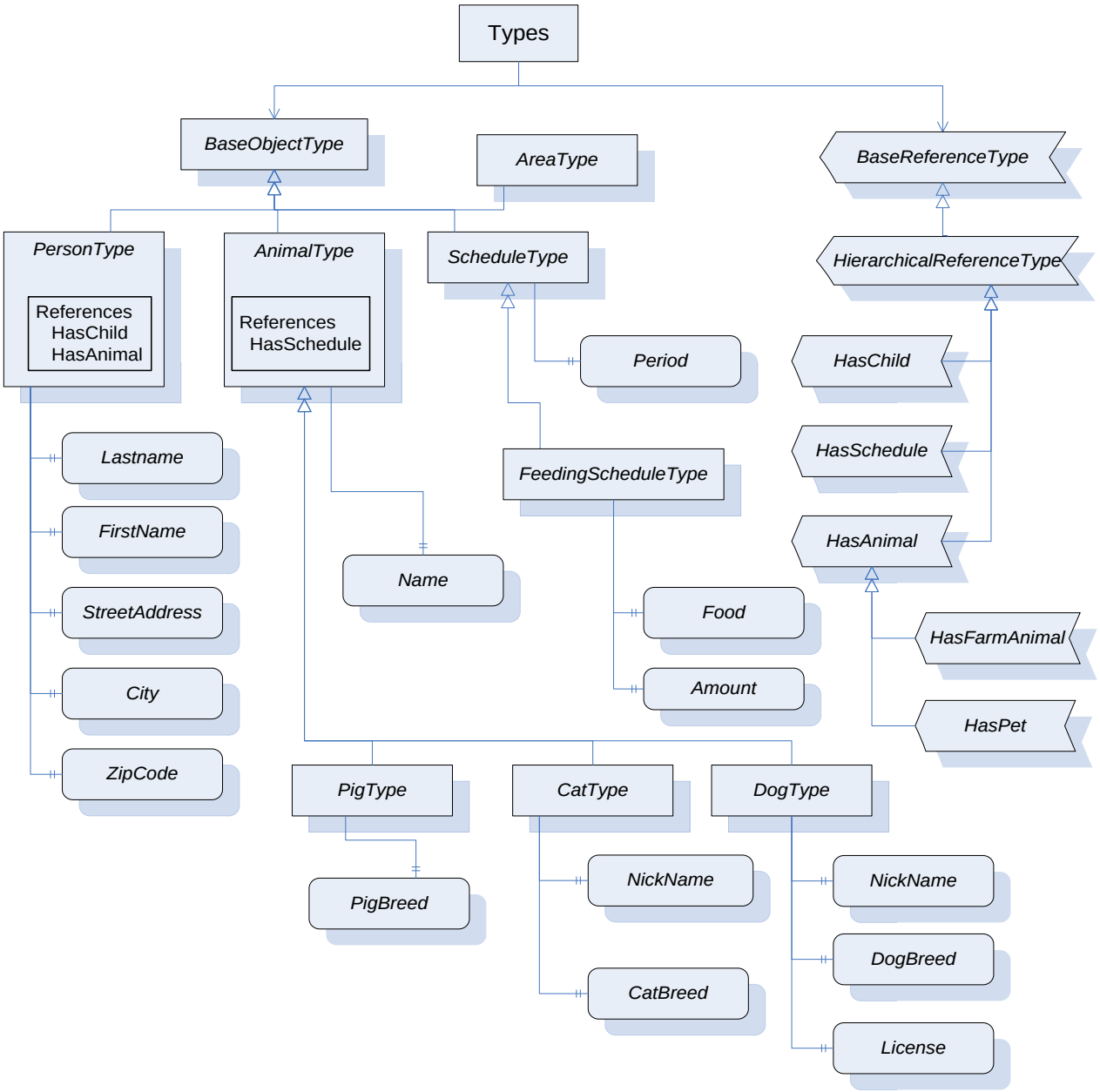


Figure B.3 – Exemples de nœuds Type

Un exemple correspondant de jeu d'instances est montré à la Figure B.4. Ces instances comprennent des *Référence* de type pour des *Objets*. Les Propriétés ont aussi des *Références* de type mais les *Références* sont omis pour la simplicité. Le nom de l'*Objet* est fourni dans la boîte et une instance numérique *NodeId* entre parenthèses. Les types *Référence* définis utilisent la notation OPC UA, les types *Référence* personnalisés sont

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

énumérés avec une *Référence* nommée. Pour les *Propriétés*, le *BrowseName*, le *NodeId* et la *Value* sont montrés. Les Noeuds qui sont inclus dans une Vue (View1) sont enfermés dans une boîte colorée. Deux nœuds Area sont inclus pour regrouper les nœuds personne existants. Tous les nœuds personnalisés sont définis dans l'espace de noms 12 qui n'est pas inclus dans la Figure B.4.

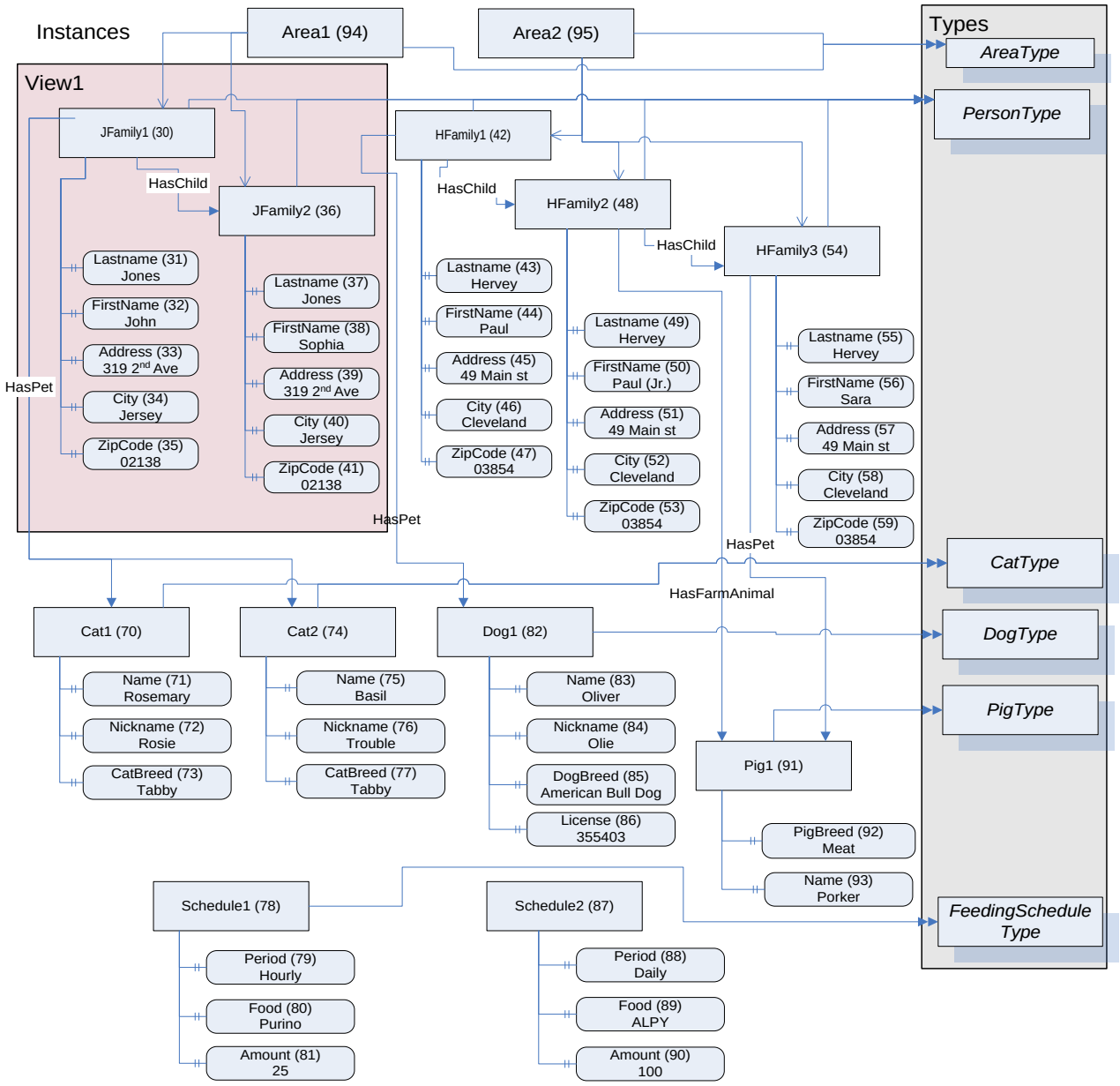


Figure B.4 – Exemple de nœuds Instance

B.2.3 Remarques relatives aux exemples

Pour tous les exemples en 7.4.4, la définition de type *Noeud* est énumérée dans sa forme symbolique; dans l'appel réel ce serait serait le *NodeId* affecté au *Noeud*. L'*AttributeId* est aussi le nom symbolique de l'*Attribut*; dans l'appel réel, il serait traduit en *IntegerId* de l'*Attribut*. Par ailleurs, dans tous les exemples, le *BrowseName* est inclus dans le tableau de résultats pour la clarté, mais normalement il n'est pas retourné.

Les exemples supposent que l'espace de noms 12 est l'espace de noms pour toutes les définitions personnalisées décrites pour les exemples.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.2.4 Exemple 1

Cet exemple illustre un filtre de simple niveau, une personne a un animal familier et l'animal familier a un programme.

Exemple 1: Get PersonType.lastName, AnimalType.name, ScheduleType.period, la personne ayant un animal familier et l'animal familier un programme.

Les paramètres *NodeTypeDescription* utilisés dans l'exemple sont décrits au Tableau B.3.

Tableau B.3 – Exemple 1 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".12:LastName"	value	N/A
		"<12:HasPet>12:AnimalType. 12:name"	value	N/A
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule. 12:period"	value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.5.

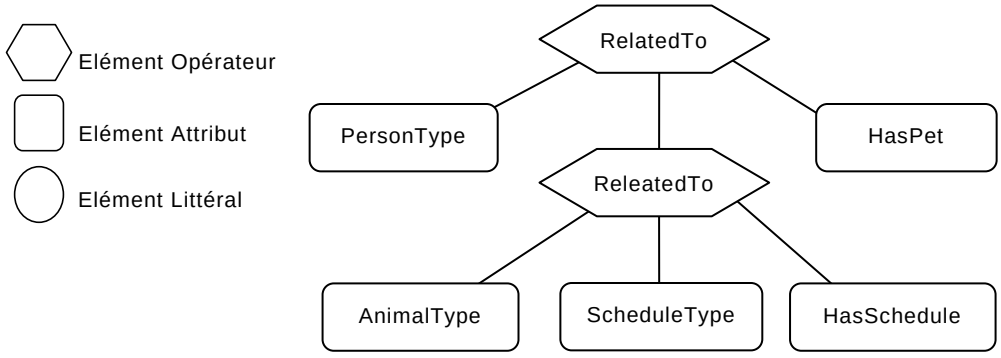


Figure B.5 – Filtre de l'exemple 1

Le Tableau B.4 décrit les éléments, les opérateurs et les opérandes de *ContentFilter* utilisés dans l'exemple.

Tableau B.4 – ContentFiltre de l'exemple 1 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
1	RelatedTo	AttributeOperand = Nodeid: PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 2	AttributeOperand = NodeId: HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: ScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand= '1'

Le Tableau B.5 décrit le *QueryDataSet* qui résulterait de cette interrogation si elle était exécutée par rapport aux instances décrites à la Figure B.4

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau B.5 – Exemple 1 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:30 (JFamily1)	PersonType	".12:lastName"	Jones
		"<12:HasPet>12:AnimalType. 12:name"	Rosemary
			Basil
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule.12:period"	Hourly
			Hourly
12:42(HFamily1)	PersonType	".12:lastName"	Hervey
		"<12:HasPet>12:AnimalType. 12:name"	Olive
		"<12:HasPet>12:AnimalType<12:HasSchedule> 12:Schedule.12:period"	Daily (tous les jours)

La colonne Valeur est retournée comme une matrice pour chaque description de Noeud, où l'ordre des éléments dans la matrice correspond à l'ordre des éléments qui étaient demandés pour le Type de Nœud donné. De plus, si un *Attribut* seul a plusieurs valeurs, il est retourné sous forme d'une matrice à l'intérieur de la matrice la plus grande. Par exemple, dans le tableau, Rosemary et Basil seraient retournés dans une matrice l'élément .<hasPet>.AnimalType.name. Ils sont montrés sur des lignes séparées pour faciliter la visualisation.

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés ici que par souci de clarté. Le *TypeDefinition NodeId* serait un nombre entier et non le nom symbolique qui est inclus dans le tableau.

B.2.5 Exemple 2

Le deuxième exemple illustre la réception d'une liste de Noeuds disjoints et montre aussi qu'une matrice de résultats peut être reçue.

Exemple 2: Get PersonType.lastName, AnimalType.name, où une personne a un enfant ou (un animal familier est du type chat et a un programme d'alimentation).

Les paramètres NodeTypeDescription utilisés dans l'exemple sont décrits au Tableau B.6.

Tableau B.6 – Exemple 2 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".12:LastName"	Value	N/A
AnimalType	TRUE (vrai)	".12:name"	Value	N/A

Le ContentFilter correspondant est illustré à la Figure B.6.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

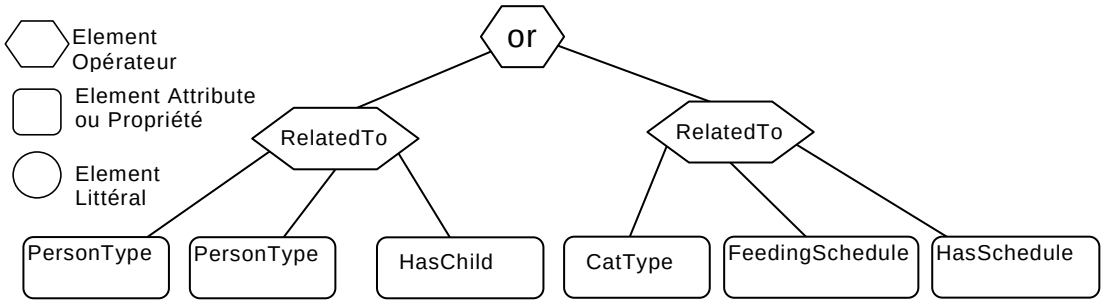


Figure B.6 – Exemple 2 Arbre logique d'un filtre

Le Tableau B.7 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple. Il est à noter qu'un Cattype est un sous-type d'Animaltype.

Tableau B.7 – Exemple 2 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

Les résultats de cette interrogation contiendraient les *QueryDataSets* présentés au Tableau B.8.

Tableau B.8 – Exemple 2 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:30 (Jfamily1)	PersonType	. 12:lastName	Jones
12:42 (HFamily1)	PersonType	. 12:lastName	Hervey
12:48 (HFamily2)	PersonType	. 12:lastName	Hervey
12:70 (Cat1)	CatType	. 12:name	Rosemary
12:74 (Cat2)	CatType	. 12:name	Basil

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés ici que par souci de clarté. Le *TypeDefinitionNodeId* serait un *NodeId* et non le nom symbolique qui est inclus dans le tableau.

B.2.6 Exemple 3

Le troisième exemple fournit une *Query* plus complexe dans laquelle les résultats sont filtrés suivant plusieurs critères.

Exemple 3: Get PersonType.lastName, AnimalType.name, ScheduleType.period où une personne a un animal familier et l'animal a un programme d'alimentation et la personne a un code postal zipcode = '02138' et la périodicité du programme est tous les jours ou toutes les heures et l'Amount (la quantité) à alimenter est > 10.

Le Tableau B.9 décrit les paramètres NodeTypeDescription utilisés dans l'exemple.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau B.9 – Exemple 3 – NodeTypeDescriptions

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d' indices
PersonType	FALSE (faux)	“12:PersonType.12:lastName”	Value	N/A
		“12:PersonType<12:HasPet>12:AnimalType. 12:name”	Value	N/A
		“12:PersonType<12:HasPet>12:AnimalType<12:HasSchedule> 12:FeedingSchedule.period”	Value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.7.

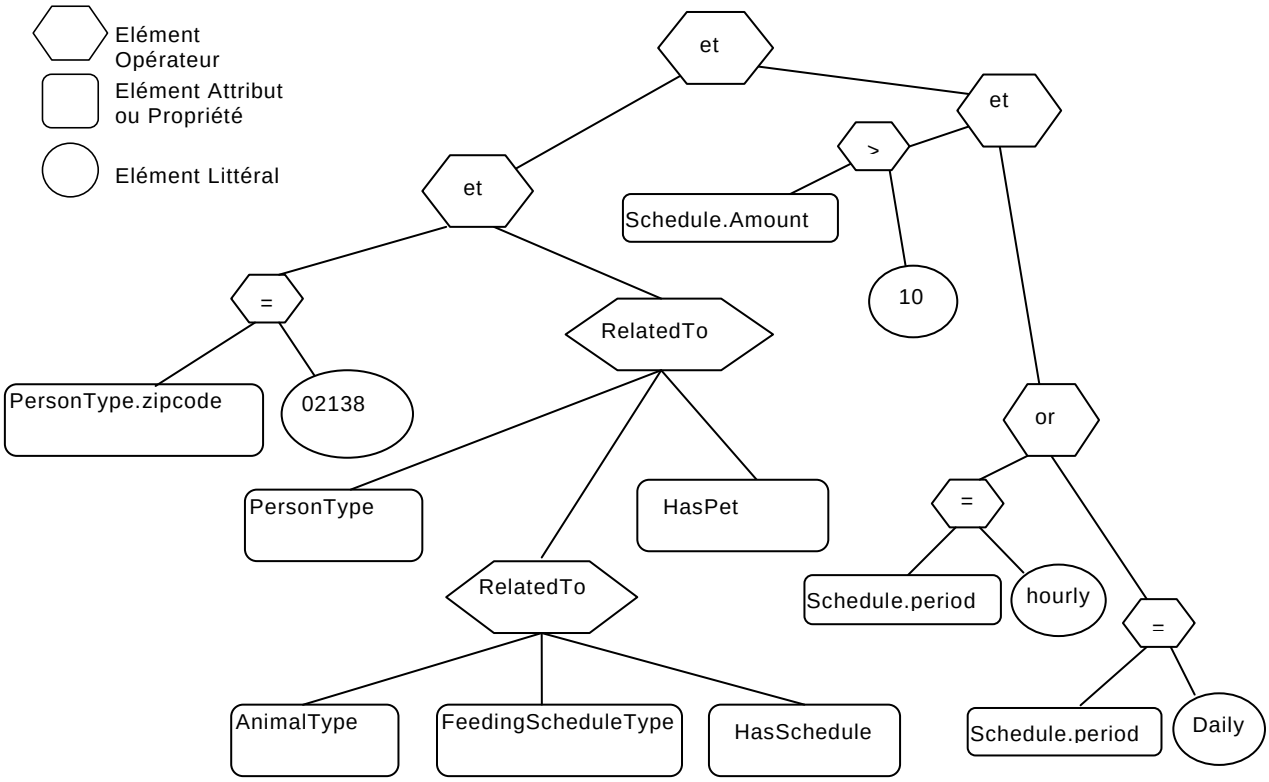


Figure B.7 – Exemple 3 Arbre logique d’un filtre

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le Tableau B.10 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.10 – Exemple 3 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	et	ElementOperand= 1	ElementOperand = 2		
1	et	ElementOperand = 3	ElementOperand = 5		
2	et	ElementOperand = 3	ElementOperand = 9		
3	Or	ElementOperand = 7	ElementOperand = 8		
4	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 5	AttributeOperand = NodeId: 12:HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
5	RelatedTo	AttributeOperand = Node: 12:AnilmalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
6	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath ".", AttributeId: zipcode	LiteralOperand = '02138'		
7	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Period	LiteralOperand = 'Daily'		
8	Equals	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Period	LiteralOperand = 'Hourly'		
9	Greater Than	AttributeOperand = NodeId: 12:PersonType BrowsePath "12:HasPet>12:AnimalType<12: HasSchedule>12: FeedingSchedule", AttributeId: Amount	ElementOperand = 10		
10	Cast	LiteralOperand = 10	AttributeOperand = NodeId: String, BrowsePath ".", AttributeId: NodeIdt		

Les résultats de cette interrogation contiendraient les *QueryDataSets* montrés au Tableau B.11.

Tableau B.11 – Exemple 3 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:30 (JFamily1)	PersonType	".12:lastName"	Jones
		"<12:hasPet>12:PersonType. 12:name"	Rosemary
			Basil
		"<12:hasPet>12:AnimalType<12:hasSchedule>12:FeedingSchedule. 12:period"	Hourly
			Hourly

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés que par souci de clarté. Le TypeDefinitionNodeId serait un nombre entier et non le nom symbolique qui est inclus dans le tableau.

B.2.7 Exemple 4

Le quatrième exemple fournit une illustration du paramètre Hop (saut) qui fait partie de l'opérateur RelatedTo.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Exemple 4: Get PersonType.lastName, avec une personne ayant un enfant qui a un enfant qui a un animal familier.

Le Tableau B.12 décrit les paramètres NodeTypeDescription utilisés dans l'exemple.

Tableau B.12 – Exemple 4 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".12:lastName"	value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.8.

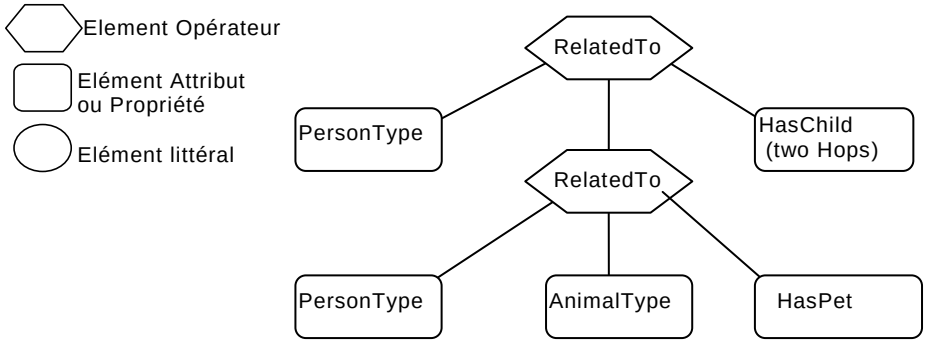


Figure B.8 – Exemple 4 Arbre logique d'un filtre

Le Tableau B.13 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.13 – Exemple 4 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand= 1	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '2'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasPet, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

Les résultats de cette interrogation contiendraient les *QueryDataSets* montrés au Tableau B.14. Il est à noter que le cochon "Pig1" est référencé comme étant un animal familier par Sara, mais est référencé comme animal de ferme par Paul le père de Sara.

Tableau B.14 – Exemple 4 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:42 (HFamily1)	PersonType	".12:lastName"	Hervey

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés ici que par souci de clarté. Le TypeDefinitionNodeId serait un nombre entier et non le nom symbolique qui est inclus dans le tableau.

B.2.8 Exemple 5

Le cinquième exemple fournit une illustration de l'utilisation d'un pseudonyme (alias).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Exemple 5: Récupérer les noms de famille d'enfants qui ont le même prénom qu'un de leurs parents.

Le Tableau B.15 décrit les paramètres NodeTypeDescription utilisés dans l'exemple

Tableau B.15 – Exemple 5 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	“<12:HasChild>12:PersonType. 12:lastName”	Value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.9.

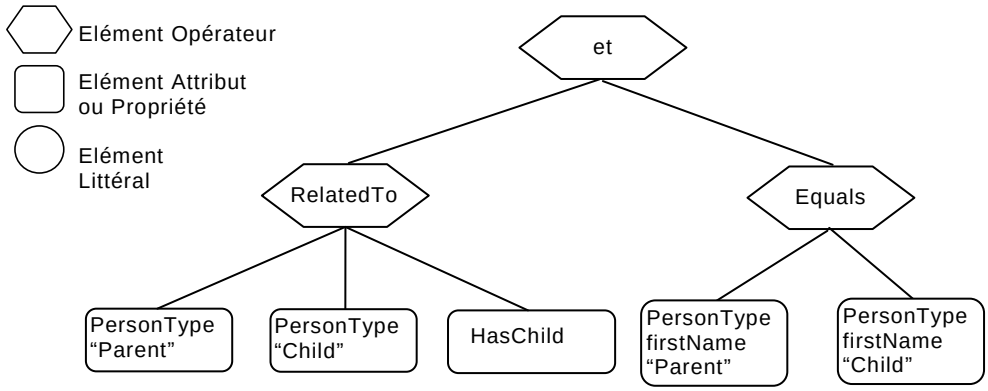


Figure B.9 – Exemple 5 Arbre logique d’une filtre

Dans cet exemple, une *Référence* à PersonType a le pseudonyme “Parent” et une autre *Référence* à PersonType a le pseudonyme “Child”. La valeur de Parent.firstName et celle de Child.firstName sont alors comparées. Le Tableau B.16 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.16 – Exemple 5 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	et	ElementOperand = 1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: NodeId, Alias: “Parent”	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: NodeId, Alias: “Child”	AttributeOperand = NodeId: 12:HasChild, AttributeId: NodeId	LiteralOperand = “1”
2	Equals	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: FirstName, Alias: “Parent”	AttributeOperand = NodeId: 12:PersonType, BrowsePath “.”, AttributeId: firstName, Alias: “Child”		

Les résultats de cette interrogation contiendraient les *QueryDataSets* montrés au Tableau B.17.

Tableau B.17 – Exemple 5 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:42 (HFamily1)	PersonType	“<12:HasChild>12:PersonType.12:lastName”	Hervey

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.2.9 Exemple 6

Le sixième exemple montre une illustration d'un type de demande différent, dans lequel le Client veut afficher une partie de l'Espace d'Adresses du serveur. Cette demande impose d'énumérer une *Référence* comme quelque chose à retourner.

Exemple 6: Get PersonType.NodeId, AnimalType.NodeId, PersonType.HasChild Reference, PersonType.HasAnimal Reference, avec une personne ayant un enfant qui a un Animal.

Le Tableau B.18 décrit les paramètres NodeTypeDescription utilisés dans l'exemple.

Tableau B.18 – Exemple 6 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".12:NodeId"	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>12:AnimalType.NodeId	value	N/A
		<12:HasChild>	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>	value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.10.

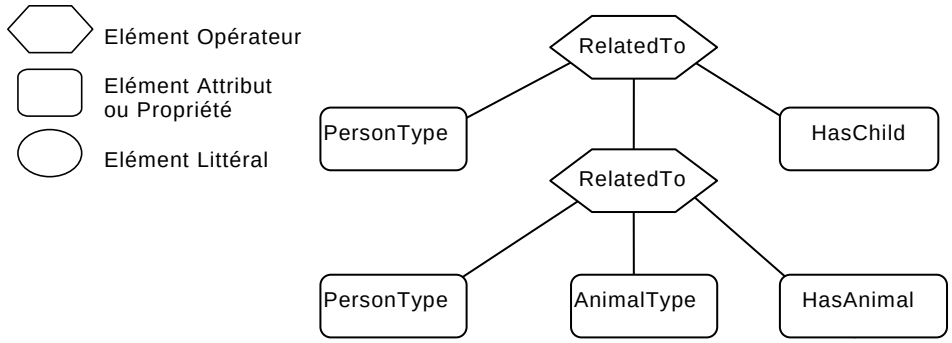


Figure B.10 – Exemple 6 Arbre logique d'un filtre

Le Tableau B.19 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.19 – Exemple 6 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 1	AttributeOperand = Node: 12:HasChild, BrowsePath ".",AttributeId:NodeId	LiteralOperand = '1'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:AnimalType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasAnimal, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

Les résultats de cette interrogation contiendraient les *QueryDataSets* montrés au Tableau B.20.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau B.20 – Exemple 6 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:42 (HFamily1)	PersonType	".NodeId"	12:42 (HFamily1)
		<12:HasChild>12:PersonType<12:HasAnimal> 12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasChild>12:PersonType<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*
12:48 (HFamily2)	PersonType	".NodeId"	12:48 (HFamily2)
		<12:HasChild>12:PersonType<12:HasAnimal> 12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasChild>12:PersonType<12:HasAnimal>	HasPet *ReferenceDescription*

NOTE Le RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses) ne sont pas dans le *QueryDataSet* et ne sont montrés que par souci de clarté, et le TypeDefinitionNodeId serait un nombre entier et non le nom symbolique qui est inclus dans le tableau. Le champ value serait dans ce cas le *NodeId* où il était demandé, mais pour l'exemple, le nom de navigation est fourni entre des parenthèses et dans le cas où des types *Références* sur le nom de navigation sont donnés. Pour les *Références* énumérés au Tableau B.20, la value (valeur) serait une *ReferenceDescription* qui est décrite en 7.24.

Le Tableau B.21 fournit un exemple du même QueryDataSet montré au Tableau B.20 sans champs supplémentaires et avec des identificateurs symboliques minimaux. Il y a une entrée pour chaque Attribut demandé; dans le cas où un Attribut retourne plusieurs entrées, les entrées sont séparées par des virgules. Si une structure est retournée, la structure est placée entre des crochets. Dans le cas d'une ReferenceDescription, la structure contient une structure. Par ailleurs, DisplayName et BrowseName sont supposés être les mêmes et définis à la Figure B.4.

Tableau B.21 – Exemple 6 QueryDataSets sans informations supplémentaires

NodeId	TypeDefinition NodeId	Valeur
12:42	PersonType	12:42
		12:91
		[HasChild,TRUE,[48,HFamily2,HFamily2,PersonType]],
		[HasFarmAnimal,TRUE[91,Pig1,Pig1,PigType]
12:48	PersonType	12:54
		12:91
		[HasChild,TRUE,[54,HFamily3,HFamily3,PersonType]]
		[HasPet, TRUE,[91,Pig1,Pig1,PigType]]

Les identificateurs PersonType, HasChild, PigType, HasPet, HasFarmAnimal utilisés dans le Tableau B.21 se traduiraient en *ExtendedNodeIds* réels.

B.2.10 Exemple 7

Le septième exemple montre une illustration d'une demande dans laquelle un *Client* souhaite afficher une partie de l'Espace d'Adresses en se basant sur un point de départ qui ont été obtenue par navigation. Cette demande inclut d'énumérer des *Références* comme quelque chose à retourner. Dans ce cas, la personne a navigué jusqu'à Area2 et a souhaité interroger *(Query*) pour obtenir des informations sous ce point de départ.

Exemple 7: Get PersonType.NodeId, AnimalType.NodeId, PersonType.HasChild Reference, PersonType.HasAnimal Reference, la personne étant dans Area2 (nœuds Cleveland) et la personne ayant un enfant.

Le Tableau B.22 décrit les paramètres NodeTypeDescription utilisés dans l'exemple.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau B.22 – Exemple 7 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".NodeId"	Value	N/A
		<12:HasChild>	Value	N/A
		<12:HasAnimal>NodeId	Value	N/A
		<12:HasAnimal>	Value	N/A

Le *ContentFilter* correspondant est illustré à la Figure B.11. Remarquer que l'appel *Browse* retournerait typiquement un *NodeId*; ainsi le premier filtre est pour le *BaseObjectType* avec un *NodeId* de 95, où 95 est le *NodeId* associé au nœud Area2, tous les Noeuds descendent de *BaseObjectType*, et *NodeId* est une *Propriété* de base et, donc, ce filtre fonctionnera pour toutes les *Queries* de cette nature.

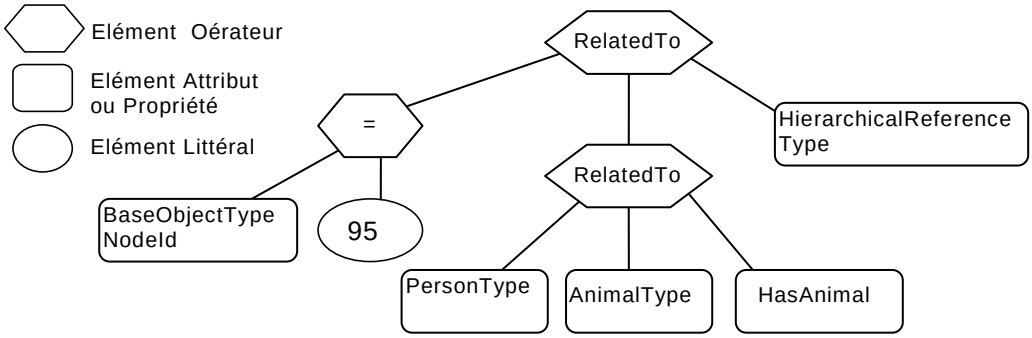


Figure B.11 – Exemple 7 Arbre logique d'un filtre

Le Tableau B.23 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.23 – Exemple 7 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	RelatedTo	AttributeOperand = NodeId: BaseObjectType, BrowsePath ".", AttributeId: NodeId	ElementOperand = 1	AttributeOperand = Node:HierachicalReference, BrowsePath ".", AttributeId:NodeId	LiteralOperand = '1'
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonTyp, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	Equals	AttributeOperand = NodeId: BaseObjectType, BrowsePath ".", AttributeId: NodeId,	LiteralOperand = '95		

Les résultats de cette *Query* contiendraient les *QueryDataSets* montrés au Tableau B.24.

Tableau B.24 – Exemple 7 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:42 (HFamily1)	PersonType	".NodeId"	12:42 (HFamily1)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasAnimal>12:AnimalType.NodeId	NULL
		<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*
12:48 (HFamily2)	PersonType	".NodeId"	12:48 (HFamily2)
		<12:HasChild>	HasChild *ReferenceDescription*
		<12:HasAnimal>12:AnimalType.NodeId	12:91 (Pig1)
		<12:HasAnimal>	HasFarmAnimal *ReferenceDescription*

NOTE Le RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses) ne sont pas dans le *QueryDataSet* et ne sont montrés que par souci de clarté et le TypeDefinitionNodeId serait un nombre entier et non

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

le nom symbolique qui est inclus dans le tableau. Le champ value serait dans ce cas le *NodeId* où il était demandé, mais pour l'exemple, le nom de navigation est fourni entre des parenthèses et dans le cas où des types *Références* sur le nom de navigation sont donnés. Pour les *Références* énumérés au Tableau B.24, la value (valeur) serait une *ReferenceDescription* qui est décrite en 7.24.

B.2.11 Exemple 8

Le huitième exemple montre une illustration d'une demande dans laquelle *l'Espace d'Adresses* est limité par une *Vue* définie par un *Serveur*. Cette demande est la même que dans le deuxième exemple qui illustre la réception d'une liste *de Noeuds* disjoints et montre aussi qu'une matrice de résultats peut être reçue. Il est **important** de noter que tous les paramètres et le *contentFilter* sont les mêmes, seule la description de la *Vue* serait spécifiée comme étant "View1".

Exemple 8: Get PersonType.lastName, AnimalType.name, où une personne a un enfant ou (un animal familier est du type chat et a un programme d'alimentation) limité par l'Espace d'Adresses dans View1.

Les paramètres NodeTypeDescription utilisés dans l'exemple sont décrits au Tableau B.25.

Tableau B.25 – Exemple 8 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".12:LastName"	value	N/A
AnimalType	TRUE (vrai)	".name"	value	N/A

Le ContentFilter correspondant est illustré à la Figure B.12.

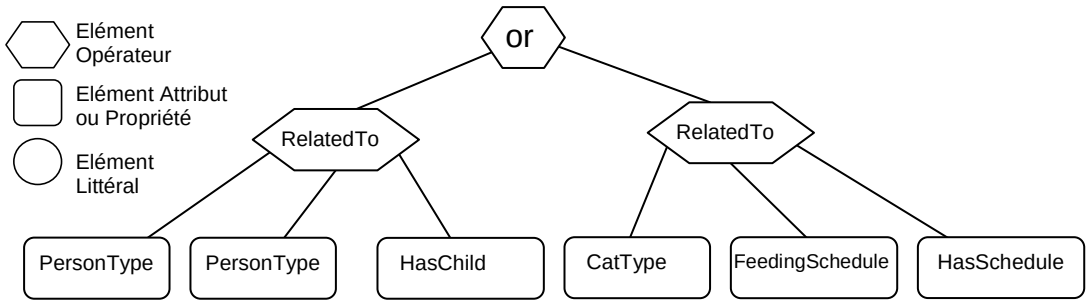


Figure B.12 – Exemple 8 Arbre logique d'un filtre

Le Tableau B.26 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple. Il est à noter qu'un CatType est un sous-type d'Animaltype.

Tableau B.26 – Exemple 8 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: 12:CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les résultats de cette interrogation contiendraient les *QueryDataSets* montrés au Tableau B.27. Si ces résultats sont comparés au jeu de résultats de l'exemple 2, la seule différence est l'omission des *Noeuds* Cat. Ces Noeuds ne sont pas dans la *Vue* et ne sont donc pas inclus dans le jeu de résultats.

Tableau B.27 – Exemple 8 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:30 (Jfamily1)	Persontype	.12:LastName	Jones

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés ici que par souci de clarté. Le TypeDefinitionNodeId serait un nombre entier et non le nom symbolique qui est inclus dans le tableau.

B.2.12 Exemple 9

Le neuvième exemple montre une autre illustration d'une demande dans laquelle *l'Espace d'Adresses* est limité par une *Vue* définie par un *Serveur*. Cette demande est semblable au deuxième exemple, excepté que certains des nœuds demandés sont exprimés en termes d'un chemin relatif. Il est **important** de remarquer que le *contentFilter* est le même, seule la description de la *Vue* serait spécifiée comme étant "View1".

Exemple 9: Get PersonType.lastName, AnimalType.name, où une personne a un enfant ou (un animal familier est du type chat et a un programme d'alimentation) limité par l'Espace d'Adresses dans View1.

Le Tableau B.28 décrit les paramètres NodeTypeDescription utilisés dans l'exemple.

Tableau B.28 – Exemple 9 NodeTypeDescription

Type Definition Node	Include Subtypes	RelativePath	ID d'attribut	Plage d'indices
PersonType	FALSE (faux)	".NodeId"	value	N/A
		<12:HasChild>12:PersonType<12:HasAnimal>12:AnimalType.NodeId	value	N/A
		<12:HasChild>	value	N/A
		<12:HasChild>12:PersonType <12:HasAnimal>	value	N/A
PersonType	FALSE (faux)	".12:LastName"	value	N/A
		<12:HasAnimal>12:AnimalType. 12:Name	value	N/A
AnimalType	TRUE (vrai)	".12:name"	value	N/A

Le ContentFilter correspondant est illustré à la Figure B.13.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

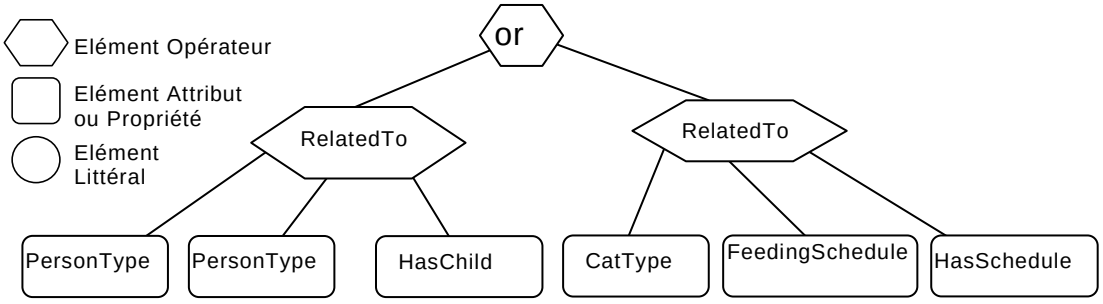


Figure B.13 – Exemple 9 Arbre logique d'un filtre

Le Tableau B.29 décrit les éléments, les opérateurs et les opérandes utilisés dans l'exemple.

Tableau B.29 – Exemple 9 ContentFilter

Element []	Opérateur	Operand[0]	Operand[1]	Operand[2]	Operand[3]
0	Or	ElementOperand=1	ElementOperand = 2		
1	RelatedTo	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:PersonType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasChild, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'
2	RelatedTo	AttributeOperand = NodeId: 12:CatType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:FeedingScheduleType, BrowsePath ".", AttributeId: NodeId	AttributeOperand = NodeId: 12:HasSchedule, BrowsePath ".", AttributeId: NodeId	LiteralOperand = '1'

Les résultats de cette interrogation (*Query*) contiendraient les *QueryDataSets* montrés au Tableau B.30. Si ces résultats sont comparés au jeu de résultats de l'exemple 2, les *Noeuds* Pet sont inclus dans la liste, même s'ils sont à l'extérieur de la *Vue*. Cela est possible car le nom référencé via le chemin relatif et le *Noeud* racine se trouvent dans la *Vue.*

Tableau B.30 – Exemple 9 QueryDataSets

NodeId	TypeDefinition NodeId	RelativePath	Valeur
12:30 (Jfamily1)	Persontype	. 12:LastName	Jones
		<12:HasAnimal>12:AnimalType. 12:Name	Rosemary
		<12:HasAnimal>12:AnimalType. 12:Name	Basil

NOTE La colonne RelativePath (chemin relatif) et le nom de navigation (entre des parenthèses dans la colonne *NodeId*) ne sont pas dans le QueryDataSet et ne sont montrés ici que par souci de clarté. Le TypeDefinitionNodeId serait un nombre entier et non le nom symbolique qui est inclus dans le tableau.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Bibliographie

CEI 62541-11, *Architecture unifiée OPC – Partie 11: Historical Access*[5]

CEI 62541-12, *Architecture unifiée OPC – Partie 12: Discovery*[6]

CEI 62541-13, *Architecture unifiée OPC – Partie 13: Agrégats*[7]

_____

5 A l'étude. Le titre français de la partie 11 n'est pas encore disponible.

6 A l'étude. Le titre français de la partie 12 n'est pas encore disponible.

7 A l'étude. Le titre français de la partie 12 n'est pas encore disponible.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-28-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe